SYBASE iAnywhere

UltraLite®
.NET Programming

Contents

About this book

This book describes UltraLite.NET™. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.

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.NET

UltraLite and .NET

UltraLite.NET applications can be deployed to Windows Mobile and Windows. If you are deploying to Windows Mobile, UltraLite.NET requires the .NET Compact Framework. If deploying to Windows, it requires the .NET Framework. UltraLite.NET also supports ActiveSync synchronization.

The .NET Compact Framework is the Microsoft .NET runtime component for Windows Mobile. It supports several programming languages. You can use either Visual Basic.NET or C# to build applications using UltraLite.NET.

UltraLite.NET provides the following namespace:

- **iAnywhere.Data.UltraLite** This namespace provides an ADO.NET interface to UltraLite. It has the advantage of being built on an industry-standard model and providing a migration path to the SQL Anywhere ADO.NET interface, which is very similar.

iAnywhere.Data.UltraLite namespace (.NET 2.0)

This chapter describes the API for the UltraLite.NET Data Provider for .NET Framework 2.0 and .NET Compact Framework 2.0.

UltraLite.NET extensions that are not available in the SQL Anywhere Data Provider for ADO.NET are denoted in this API reference with **UL Ext.:**.

To use the UltraLite Engine runtime of UltraLite.NET, set the Runtime Type property to the appropriate value before using any other UltraLite.NET API. See "RuntimeType property" on page 213.

Applications must open a connection to perform operations on a database. Connections are opened using the "ULConnection class" on page 110.

The **iAnywhere.Data.UltraLite** assembly uses a satellite resource assembly called **iAnywhere.Data.UltraLite.resources**. The main assembly searches for this resource assembly by culture, using the following order:

- CultureInfo.CurrentUICulture: http://msdn2.microsoft.com/en-us/library/System.Globalization.CultureInfo.CurrentUICulture.aspx
- CultureInfo.CurrentCulture: http://msdn2.microsoft.com/en-us/library/System.Globalization.CultureInfo.CurrentCulture.aspx
- **EN**

Many of the properties and methods in this chapter are very similar to the .NET Framework Data Provider for OLE DB (System.Data.OleDb). You can find more information and examples in the Microsoft .NET Framework documentation.

System requirements and supported platforms

Development platforms

To develop applications using UltraLite.NET, you require the following:

- A supported desktop version of Microsoft Windows.
- Microsoft Visual Studio 2005 or Visual Studio 2008.
- For Windows Mobile devices, .NET Compact Framework version 2 or later.

Target platforms

UltraLite.NET supports the following target platforms:

- Microsoft .NET Compact Framework 2.0 and .NET Framework 2.0.
- For Windows Mobile devices, Microsoft .NET Compact Framework version 2 or later.
- Microsoft .NET Compact Framework version 2 or later on Windows.

In addition to your application code and the .NET Compact Framework, you must deploy the following files to your Windows Mobile device or computer running Windows:

- **iAnywhere.Data.UltraLite.dll** An assembly containing the iAnywhere.Data.UltraLite ADO.NET namespace. This file is required only if your application uses the iAnywhere.Data.UltraLite namespace.
- **iAnywhere.Data.UltraLite.resources.dll** The resources needed by the iAnywhere.Data.UltraLite ADO.NET namespace. This file is required only if your application uses the iAnywhere.Data.UltraLite namespace.
- **ulnet12.dll** This file contains the UltraLite runtime used by a single client. A separate version of the runtime is provided for each target platform.
- **ulnetclient12.dll** This file contains the interface to the UltraLite engine and is used to allow multiple clients to access the engine.

For information about UltraLite supported platforms, see http://www.sybase.com/detail?id=1002288.

UltraLite.NET architecture

The UltraLite.NET namespace is named iAnywhere.Data.UltraLite (ADO.NET interface).

The following list describes some of the more commonly-used high level classes for the iAnywhere.Data.UltraLite ADO.NET namespace:

- **ULConnection** Each ULConnection object represents a connection to an UltraLite database. You can create one or more ULConnection objects.
- **ULTable** Each ULTable object provides access to the data in a single table.
- **ULCommand object** Each ULCommand object holds a SQL statement to be executed against the database.
- **ULDataReader object** Each ULDataReader object holds the result set for a single query.
- **ULSyncParms** You use the ULSyncParms object to synchronize your UltraLite database with a MobiLink server.

For more information about the ADO.NET interface, see "UltraLite.NET API reference" on page 37.

Understanding UltraLite.NET development

Using SQL Anywhere tools in Visual Studio

Some SQL Anywhere tools can be accessed from Microsoft Visual Studio. Sybase Central and Interactive SQL are available from the Visual Studio **Tools** menu.

Connecting to a database

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

Using the ULConnection object

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

- **Commit behavior** By default, UltraLite.NET applications are in AutoCommit mode. Each Insert, Update, or Delete statement is committed to the database immediately. You can use ULConnection.BeginTransaction to define the start of a transaction in your application. See "Managing transactions" on page 16.
- **User authentication** You can change the user ID and password for the application from the default values of DBA and sql by using methods to grant or revoke connection permissions. Each UltraLite database can define a maximum of four user IDs. See "Authenticating users" on page 18.
- **Synchronization** A set of objects governing synchronization is accessed from the Connection object. See "Synchronization in UltraLite applications" on page 19.
- **Tables** UltraLite tables are accessed using methods of the Connection object. See "Accessing and manipulating data with the Table API" on page 10.
- **Commands** A set of objects is provided to handle the execution of dynamic SQL statements and to navigate result sets. See "Accessing and manipulating data using SQL" on page 7.

See "ULConnection class" on page 110.

Multi-threaded applications

Each ULConnection and all objects created from it should be used on a single thread. If your application requires multiple threads accessing the UltraLite database, each thread requires a separate connection. For example, if you design your application to perform synchronization in a separate thread, you must use a separate connection for the synchronization and you must open the connection from that thread.

To connect to an UltraLite database

1. Declare a ULConnection object.

   Most applications use a single connection to an UltraLite database and leave the connection open. Multiple connections are only required for multi-threaded data access. For this reason, it is often best to declare the ULConnection object as global to the application.

   ```
   ULConnection conn;
   ```

2. Open a connection to an existing database.

   UltraLite applications must deploy an initial database file or the application must include code to create the database file. The initial database file can be created by using Sybase Central or the command line utilities provided with UltraLite.

   You can specify connection parameters either as a connection string or using the ULConnectionParms object. The following example illustrates using the ULConnectionParms object to connect to an UltraLite database named*mydata.udb*.

   ```
   ULConnectionParms parms = new ULConnectionParms();
   parms.DatabaseOnDesktop = "mydata.udb";
   conn = new ULConnection( parms.ToString() );
   conn.Open();
   ```

Code samples in C#
The code samples in this chapter are in Microsoft C#. If you are using one of the other supported development tools, you must modify the instructions appropriately.

Encryption and obfuscation

By default, the data in a new UltraLite database is not encrypted. By specifying appropriate database creation parameters, the database may be created with strong encryption or with simple obfuscation. Obfuscation is a very weak form of keyless encryption that is only intended to prevent casual observation of the data in the database (with a low-level file examination utility for example).

Encryption

To create a database with strong encryption, specify an encryption key when creating the database using Sybase Central or specify the encryption key in the creation parameters if the database is created by calling ULCreateDatabase or using the ulinit utility. To be effective, the encryption key should contain a combination of characters, numbers, and special symbols. Using a long encryption key reduces the chances of someone guessing the key.

Once a database is encrypted, the encryption key cannot be recovered. Access to the database is completely lost unless the proper encryption key is specified. Encryption keys should be treated as sensitive information and archived appropriately.

See “UltraLite DBKEY connection parameter” [*UltraLite - Database Management and Reference*].

You can change the encryption key for an existing UltraLite database by applying a new encryption key with the Connection.ChangeEncryptionKey method.

See "ULConnection class" on page 110 and "ULConnectionParms class" on page 154.

After the database is encrypted, connections to the database must specify the correct encryption key; otherwise, the connections fail.

Obfuscation

To obfuscate the database, specify **obfuscate=y** as a database creation parameter. For more information about database encryption and obfuscation parameters, see "Choosing database creation parameters for UltraLite" [*UltraLite - Database Management and Reference*].

Accessing and manipulating data using SQL

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

For information about using the Table API, see "Accessing and manipulating data with the Table API" on page 10.

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

- Inserting, deleting, and updating rows.
- Executing queries and retrieving rows to a result set.
- 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 statements" [*SQL Anywhere Server - SQL Reference*].

Data manipulation: INSERT, UPDATE, and DELETE

With UltraLite, you can perform SQL data manipulation language operations. These operations are performed using the ULCommand.ExecuteNonQuery method.

See "ULCommand class" on page 64.

Placeholders for parameters in SQL statements are indicated by the ? character. For any INSERT, UPDATE, or DELETE, each ? is referenced according to its ordinal position in the command's parameters collection. For example, the first ? is referred to as 0, and the second as 1.

To insert a row

1. Declare a ULCommand.

   ```
   ULCommand cmd;
   ```

2. Assign a SQL statement to the ULCommand object.

   ```
   cmd = conn.CreateCommand();
   cmd.Command = "INSERT INTO MyTable(MyColumn) values (?)";
   ```

3. Assign input parameter values for the statement.

   The following code shows a string parameter.

   ```
   String newValue;
   // assign value
   cmd.Parameters.add("", newValue);
   ```

4. Execute the statement.

   The return value indicates the number of rows affected by the statement.

   ```
   int rowsInserted = cmd.ExecuteNonQuery();
   ```

5. If you are using explicit transactions, commit the change.

   ```
   myTransaction.Commit();
   ```

To update a row

1. Declare a ULCommand.

   ```
   ULCommand cmd;
   ```

2. Assign a statement to the ULCommand object.

   ```
   cmd = conn.CreateCommand();
   cmd.Command = "UPDATE MyTable SET MyColumn1 = ? WHERE MyColumn2 = ?";
   ```

3. Assign input parameter values for the statement.

   ```
   String newValue;
   String oldValue;
   // assign values
   cmd.Parameters.add("", newValue);
   cmd.Parameters.add("", oldValue);
   ```

4. Execute the statement.

   ```
   int rowsUpdated = cmd.ExecuteNonQuery();
   ```

5. If you are using explicit transactions, commit the change.

   ```
   myTransaction.Commit();
   ```


To delete a row

1. Declare a ULCommand.

```
ULCommand cmd;
```

2. Assign a statement to the ULCommand object.

```
cmd = conn.CreateCommand();
cmd.Command = "DELETE FROM MyTable WHERE MyColumn = ?";
```

3. Assign input parameter values for the statement.

```
String deleteValue;
// assign value
cmd.Parameters.add("", deleteValue);
```

4. Execute the statement.

```
int rowsDeleted = cmd.ExecuteNonQuery();
```

5. If you are using explicit transactions, commit the change.

```
myTransaction.Commit();
```

Data retrieval: SELECT

The SELECT statement allows you to retrieve information from the database. This section describes how to execute a SELECT statement and how to handle the result set it returns.

To execute a SELECT statement

1. Declare a ULCommand object, which holds the query.

```
ULCommand cmd;
```

2. Assign a statement to the object.

```
cmd = conn.CreateCommand();
cmd.Command = "SELECT MyColumn FROM MyTable";
```

3. Execute the statement.

   Query results can be returned as one of several types of objects. In this example, a ULDataReader object is used. In the following code, the result of the SELECT statement contains a string, which is output to the command prompt.

```
ULDataReader customerNames = prepStmt.ExecuteReader();
int fc = customerNames.GetFieldCount();
while( customerNames.MoveNext() ) {
  for ( int i = 0; i < fc; i++ ) {
    System.Console.Write(customerNames.GetString( i ) + " " );
  }
```

```
        System.Console.WriteLine();
    }
```

Navigating SQL result sets

You can navigate through a result set using methods associated with the ULDataReader object.

The result set object provides you with the following methods to navigate a 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(offset)** moves a certain number of rows relative to the current row, as specified by the offset. Positive offset values move forward in the result set, relative to the current position of the cursor in the result set, and negative offset values move backward in the result set. An offset value of zero does not move the cursor, but allows you to repopulate the row buffer.

Result set schema description

The ULDataReader.GetSchemaTable method and ULDataReader.Schema property allow you to retrieve information about a result set, such as column names, total number of columns, column scales, column sizes, and column SQL types.

Example

The following example demonstrates how to use the ULDataReader.Schema and ResultSet.Schema properties to display schema information in a command prompt.

```
for ( int i = 0; i < MyResultSet.Schema.GetColumnCount(); i++ ) {
      System.Console.WriteLine( MyResultSet.Schema.GetColumnName(i)
                                + " "
                                + MyResultSet.Schema.GetColumnSQLType(i) );
}
```

Accessing and manipulating data with the Table API

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

For more information about SQL, see "Accessing and manipulating data using SQL" on page 7.

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

- Scroll through the rows of a table.
- Access the values of the current row.
- Use find and lookup methods to locate rows in a table.
- Insert, delete, and update rows.

Navigating the rows of a table

UltraLite.NET provides you with several methods to navigate a table to perform a wide range of navigation tasks.

The table object provides you with the following methods to navigate a table.

- **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(offset)** moves a certain number of rows relative to the current row, as specified by the offset. Positive offset values move forward in the table, relative to the current position of the cursor in the table, and negative offset values move backward in the table. An offset value of zero does not move the cursor, but allows you to repopulate the row buffer.

Example

The following code opens the MyTable table and displays the value of the MyColumn column for each row.

```
ULTable t = conn.ExecuteTable( "MyTable" );
int colID = t.GetOrdinal( "MyColumn" );
while ( t.MoveNext() ){
   System.Console.WriteLine( t.GetString( colID ) );
}
```

You expose the rows of the table to the application when you open the table object. By default, the rows are ordered by primary key value, but you can specify an index when opening a table to access the rows in a particular order.

Example

The following code moves to the first row of the MyTable table as ordered by the ix_col index.

```
ULTable t = conn.ExecuteTable( "MyTable", "ix_col" );
t.MoveFirst();
```

See "ULTable class" on page 394 and "ULTableSchema class" on page 417.

Using UltraLite modes

An 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 insert method is called.
- **Update mode** The data in the buffer replaces the current row when the update method is called.
- **Find mode** Used to locate a row whose value exactly matches the data in the buffer when one of the 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 lookup methods is called.

Accessing the values of the current row

A Table object is always located at one of the following positions:

- Before the first row of the table.
- On a row of the table.
- After the last row of the table.

If the Table object is positioned on a row, you can use one of a set of methods appropriate for the data type to retrieve or modify the value of each column.

Retrieving column values

The Table object provides a set of methods for retrieving column values. These methods take the column ID as argument.

Examples

The following code retrieves the value of the lname column, which is a character string.

```
int lname = t.GetOrdinal( "lname" );
string lastname = t.GetString( lname );
```

The following code retrieves the value of the cust_id column, which is an integer.


```
int cust_id = t.GetOrdinal( "cust_id" );
int id = t.GetInt( cust_id );
```

Modifying column values

In addition to the methods for retrieving values, there are methods for setting values. These methods take the column ID and the value as arguments.

Example

For example, the following code sets the value of the lname column to Kaminski.

```
t.SetString( lname, "Kaminski" );
```

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

```
// This code is incorrect
t.MoveBeforeFirst();
id = t.GetInt( cust_id );
```

Casting values

The method you choose must match the data type you want to assign. UltraLite automatically casts database data types where they are compatible, so that you could use the getString method to fetch an integer value into a string variable, and so on. See "Converting data types explicitly" [*UltraLite - Database Management and Reference*].

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 Table 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 specified search values, under the sort order specified when the Table object was opened. If the search values cannot be found, the application is positioned before the first or after the last row.
- **Lookup methods** move to the first row that matches or is greater than a specified search value, under the sort order specified when the Table object was opened.

To search for a row

1. Enter find or lookup mode.

   The mode is entered by calling a method on the table object. For example, the following code enters find mode.

```
t.FindBegin();
```

2. Set the search values.

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

   ```
   int lname = t.GetOrdinal( "lname" );
   t.SetString( lname, "Kaminski" );
   ```

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.

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

   ```
   tCustomer.FindFirst();
   ```

4. Search for the next instance of the row.

   Use the appropriate method to carry out the search. For a find operation, FindNext locates the next instance of the parameters in the index. For a lookup, MoveNext locates the next instance.

See "ULTable class" on page 394.

Updating rows

The following procedure describes how to update a row.

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 the table using find or lookup methods.

2. Enter update mode.

   For example, the following instruction enters update mode on table t.

   ```
   t.BeginUpdate();
   ```

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

   For example, the following instruction sets the id column in the buffer to 3.

   ```
   t.SetInt( id , 3);
   ```

4. Execute the Update.


```
t.Update();
```

After the update operation, the current row is the row that has been updated. If you changed the value of a column in the index specified when the Table object was opened, the current row is undefined.

By default, UltraLite.NET operates in AutoCommit mode, so that the update is immediately applied to the row in permanent storage. If you have disabled AutoCommit mode, the update is not applied until you execute a commit operation. See "Managing transactions" on page 16.

Caution
You cannot update the primary key value of a row: delete the row and add a new row instead.

Inserting rows

The steps to insert a row are very similar to those for updating rows, except that there is no need to locate a row in the table before carrying out the insert operation. The order of row insertion into the table has no significance.

Example

The following code inserts a new row.

```
t.InsertBegin();
t.SetInt( id, 3 );
t.SetString( lname, "Carlo" );
t.Insert();
```

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

- For nullable columns, NULL.
- For numeric columns that disallow NULL, zero.
- For character columns that disallow NULL, an empty string.
- To explicitly set a value to NULL, use the setDBNull method.

For update operations, an insert is applied to the database in permanent storage when a commit is carried out. In AutoCommit mode, a commit is carried out as part of the insert method.

Deleting rows

The steps to delete a row are simpler than to insert or update rows. There is no delete mode corresponding to the insert or update modes.

The following procedure deletes a row.

To delete a row

1. Move to the row you want to delete.
2. Execute the Table.Delete method.

```
t.Delete();
```

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 an entire transaction is executed, or none of the statements in the transaction are executed.

By default, UltraLite.NET operates in AutoCommit mode, so that each insert, update, or delete is executed as a separate transaction. Once the operation is complete, the change is made to the database.

To use multi-statement transactions, you must create a ULTransaction class object by calling ULConnection.BeginTransaction. For example, if your application transfers money between two accounts, both the deduction from the source account and the addition to the destination account must be completed as a distinct operation, otherwise both statements must not be completed.

If the connection has performed a valid transaction, you must execute ULTransaction.Commit statement to complete the transaction and commit the changes to your database. If the set of updates is to be abandoned, execute ULTransaction.Rollback statement to cancel and roll back all the operations of the transaction. Once a transaction has been committed or rolled back, the connection will revert to AutoCommit mode until a subsequent call to ULConnection.BeginTransaction.

For example, the following code fragment shows how to set up a transaction that involves multiple operations (avoiding the default autocommit behavior):

```
// Assuming an already open connection named  conn
ULTransaction txn = conn.BeginTransaction(IsolationLevel.ReadUncommitted);
// Carry out transaction operations here
txn.Commit();
```

UltraLite isolation level
UltraLite supports only the IsolationLevel.ReadUncommitted member of the IsolationLevel enumeration.

Some SQL statements—especially statements that alter the structure of the database—cause any pending transactions to be committed. Examples of SQL statements that automatically commit transactions in progress are: CREATE TABLE and ALTER TABLE.

See “ULConnection class” on page 110 and “ULTransaction class” on page 430.

Accessing schema information

The objects in the table API represent tables, columns, indexes, and synchronization publications. Each object has a Schema property that provides access to information about the structure of that object.

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

You can access the following schema objects and information:

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

  Call ULConnection.Schema to obtain a ULDatabaseSchema object. See "Schema property" on page 149 and "ULDatabaseSchema class" on page 213.

- **ULTableSchema** The number and names of the columns and indexes for this table.

  Call ULTable.Schema to obtain a ULTableSchema object. See "Schema property" on page 416 and "ULTableSchema class" on page 417.

- **ULIndexSchema** Information about the column in the index. As an index has no data directly associated with it there is no separate Index class, just a ULIndexSchema class.

  Call the ULTableSchema.GetIndex, ULTableSchema.GetOptimalIndex, or ULTableSchema.GetPrimaryKey method to obtain a ULIndexSchema object. See "ULTableSchema class" on page 417 and "ULIndexSchema class" on page 283.

Handling errors

You can use the standard .NET error-handling features to handle errors. Most UltraLite methods throw ULException errors. You can use ULException.NativeError to retrieve the ULSQLCode value assigned to this error. ULException has a Message property, which you can use to obtain a descriptive text of the error. ULSQLCode errors are negative numbers indicating the error type.

For a list of error codes, see "Error Messages".

After synchronization, you can use the SyncResult property of the connection to obtain more detailed error information. For example, the following sample illustrates a possible technique for reporting errors that occur during synchronization:

```
public void Sync()
  {
        try
        {
           _conn.Synchronize( this );
           _inSync = false;
        }
        catch( ULException uEx ){
           if( uEx.NativeError == ULSQLCode.SQLE_COMMUNICATIONS_ERROR )
           {
              MessageBox.Show(
                   "StreamErrorCode = " +
                    _conn.SyncResult.StreamErrorCode.ToString() +
```

```
                    "\r\n"
                + "StreamErrorContext = " +
                  _conn.SyncResult.StreamErrorContext + "\r\n"
                + "StreamErrorID = " +
                  _conn.SyncResult.StreamErrorID + "\r\n"
                + "StreamErrorSystem = " +
                  _conn.SyncResult.StreamErrorSystem + "\r\n"
                );
            }
            else
            {
                MessageBox.Show(uEx.Message);
            }
        }
        catch(System.Exception ex )
        {
            MessageBox.Show(ex.Message);
        }
    }
```

See also

- "ULSyncProgressListener interface" on page 389
- "ULSyncResult class" on page 390

Authenticating users

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

You cannot directly change a user ID. Instead, add the new user ID and delete the existing user ID. UltraLite supports a maximum of four user IDs for each UltraLite database.

See "Authenticating users" on page 18.

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

1. Connect to the database using the user ID and password of an existing user.
2. Grant user access to the database with the desired password using the ULConnection.GrantConnectTo method.

   This procedure is the same whether you are adding a new user or changing the password of an existing user.

   See "ULConnection class" on page 110.

To delete an existing user

1. Connect to the database using the user ID and password of an existing user.
2. Delete an existing user using the Connection.RevokeConnectFrom method.

Synchronization in UltraLite applications

You synchronize an UltraLite database with a central consolidated database. Synchronization requires the MobiLink synchronization software included with SQL Anywhere.

This section provides a brief introduction to synchronization and describes some features of particular interest to users of UltraLite.NET.

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

You can also find a working example of synchronization in the CustDB sample application. For more information, see the *Samples\UltraLite.NET\CustDB* subdirectory of your SQL Anywhere 12 installation.

UltraLite.NET supports TCP/IP, HTTP, HTTPS, and TLS (transport-layer security) synchronization. Synchronization is initiated by the UltraLite application. Always use properties of the SyncParms 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*].

Initiating Synchronization in C# Application

The following code illustrates how to initiate synchronization in an application written in C#.

```
private void Sync()
{
    // Sync
    try
    {
         // setup to synchronize a publication named "high_priority"
         conn.SyncParms.Publications = "high_priority";

         // Set the synchronization parameters
         conn.SyncParms.Version     = "Version1";
         conn.SyncParms.StreamParms = "";
         conn.SyncParms.Stream      = ULStreamType.TCPIP;
         conn.SyncParms.UserName    = "51";
         conn.Synchronize();
    }
    catch (System.Exception t)
         {
           MessageBox.Show("Exception: " + t.Message);
         }
}
```

Adding ActiveSync synchronization to your application

This section describes how to add ActiveSync synchronization to an UltraLite.NET application, and how to register your application for use with ActiveSync on your end users' computers.

ActiveSync synchronization can only be initiated by ActiveSync. ActiveSync initiates synchronization when the device is placed in the cradle or when **Synchronize** is selected from the ActiveSync window.

When ActiveSync initiates synchronization, the MobiLink provider for ActiveSync starts the UltraLite application, if it is not already running, and sends a message to it. Your application must implement a ULActiveSyncListener object to receive and process messages from the MobiLink provider. Your application must specify the listener object using the SetActiveSyncListener method, where **MyAppClassName** is a unique Windows class name for the application.

```
dbMgr.SetActiveSyncListener( "MyAppClassName", listener );
```

For more information, including sample code, see “ULActiveSyncListener interface” on page 37.

When UltraLite receives an ActiveSync message, it invokes the specified listener's ActiveSyncInvoked method on a different thread. To avoid multi-threading issues, your ActiveSyncInvoked method should post an event to the user interface.

If your application is multi-threaded, use a separate connection and use the **lock** keyword in C# or **SyncLock** keyword in Visual Basic .NET to access any objects shared with the rest of the application. The ActiveSyncInvoked method should specify a ULStreamType.ACTIVE_SYNC for its connection's SyncParms.Stream and then call ULConnection.Synchronize.

When registering your application, set the following parameter:

- **Class Name** The same class name the application used with the Connection.SetActiveSyncListener method.

Tutorial: Build an UltraLite.NET application

Introduction to UltraLite.NET development tutorial

This tutorial guides you through the process of building an UltraLite application using Microsoft Visual Studio. It uses the ADO.NET interface provided by the iAnywhere.Data.UltraLite namespace.

This tutorial contains code for a Visual Basic application and a Visual C# application.

Competencies and experience

This tutorial assumes the following:

- You are familiar with the C# programming language or the Visual Basic programming language.
- You have Microsoft Visual Studio installed on your computer and you are familiar with using Visual Studio. This tutorial is tested using Visual Studio 2008 and may refer to Visual Studio actions or procedures that may be slightly different in other versions of Visual Studio.
- You know how to create an UltraLite database using the UltraLite plug-in for Sybase Central.

  See "Create a database with the Create Database Wizard" [*UltraLite - Database Management and Reference*].

Goals

The goal for the tutorial is to gain competence and familiarity with the process of developing UltraLite applications in the Visual Studio environment.

Installation note

If you install UltraLite software on a Windows computer that already has Visual Studio installed, the UltraLite installation process detects the presence of Visual Studio and performs the necessary integration steps. If you install Visual Studio after installing UltraLite, or install a new version of Visual Studio, the process to integrate UltraLite with Visual Studio must be performed manually at a command prompt as follows:

- Ensure Visual Studio is not running.
- For Visual Studio 2005 or later, run *installULNet.exe* from the folder named *install-dir\UltraLite\UltraLite.NET\Assembly\v2*.

Lesson 1: Create Visual Studio project

The following procedure creates and configures a new Visual Studio application. You can choose whether to use Visual Basic or C# as your programming language.

This tutorial assumes that if you are designing a C# application, your files are in the directory *C:\tutorial\uldotnet\CSApp* and that if you are designing a Visual Basic application, your files are in the directory *C:*

\tutorial\uldotnet\VBApp. If you choose to use a directory with a different name, use that directory throughout the tutorial.

To create a Visual Studio project

1. Create a Visual Studio project.
   - From the Visual Studio **File** menu, choose **New** » **Project**.
   - The **New Project** window appears. In the left pane, expand either the **Visual Basic** folder or the **Visual C#** folder. Select **Smart Device** as the project type.

     In the right pane, select a **Smart Device Project** and name your project **VBApp** or **CSApp**, depending on whether you are using Visual Basic or C# for the programming language.
   - Enter a Location of *C:\tutorial\uldotnet* and click **OK**.
   - Choose **Windows Mobile 5.0 Pocket PC SDK** as the target platform. Click **OK**.
2. Add references to your project.
   - Add the iAnywhere.Data.UltraLite assembly and the associated resources to your project.
     a. From the **Project** menu, choose **Add Reference**.
     b. Select **iAnywhere.Data.UltraLite (CE)** from the list of available references. Click **Select** to add it to the list of selected components.

        If this reference does not appear in the list, click **Browse** and locate it in the *UltraLite \UltraLite.NET\ce\Assembly\v2* subdirectory of your SQL Anywhere installation. Select *iAnywhere.Data.UltraLite.dll* and click **OK**.
     c. Select **iAnywhere.Data.UltraLite (CE) EN** from the list of available references. Click **Select** to add it to the list of selected components.

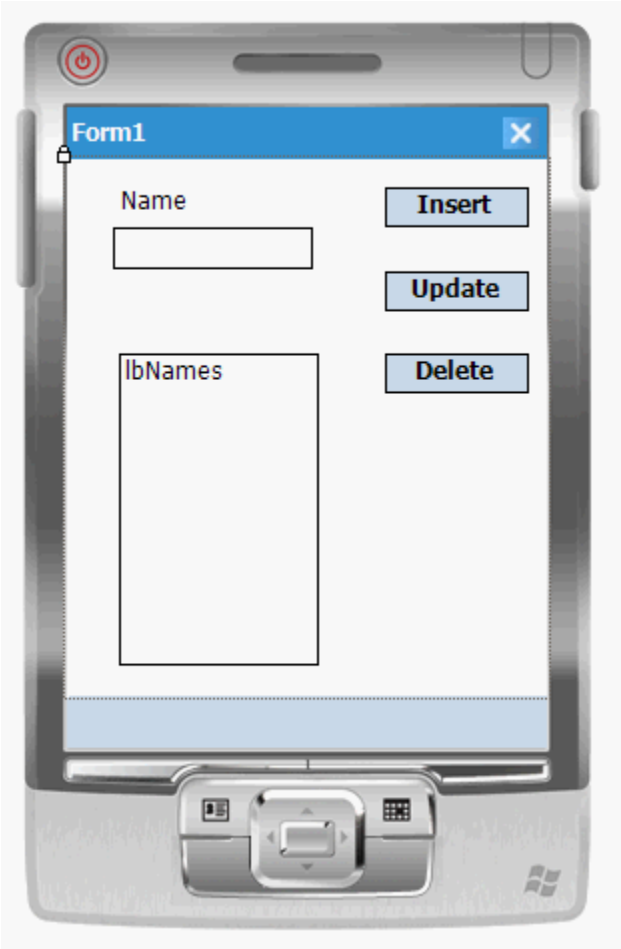
        If this reference does not appear in the list, click **Browse** and locate it in the *UltraLite \UltraLite.NET\ce\Assembly\v2\xx* subdirectory of your SQL Anywhere installation, where xx is a two-letter abbreviation for the desired language (for example, use **en** for English). Select *iAnywhere.Data.UltraLite.resources.dll* and click **Open**.
     d. Click **OK** to add the assembly and resources to your project.
   - Link the UltraLite component to your project.

     In this step, ensure that you add a link to the component, and that you do not open the component.
     a. From the **Project** menu, choose **Add Existing Item** and browse to the *UltraLite\UltraLite.NET \ce* subdirectory of your SQL Anywhere installation.
     b. In the **Files of Type** list, choose **Executable Files**.
     c. Open the folder corresponding to the processor of the Windows Mobile device you are using. For Visual Studio 2005 and later, open the *arm.50* folder. Select *ulnet12.dll*; Click the arrow on the **Add** button and select **Add as Link**.
3. Create a form for your application.

If the Visual Studio toolbox panel is not currently displayed, from the main menu choose **View** » **Toolbox**. Add the following visual components to the form by selecting the object from the toolbox and dragging it onto the form in the desired location.

Type	Design - name	Appearance - text
Button	btnInsert	Insert
Button	btnUpdate	Update
Button	btnDelete	Delete
TextBox	txtName	(no text)
ListBox	lbNames	(no text)
Label	laName	Name

Your form should look like the following figure:



4. Build and deploy your solution.

   Building and deploying the solution confirms that you have configured your Visual Studio project properly.

   a. From the **Build** menu, choose **Build Solution**. Confirm that the project builds successfully. If you are building a Visual Basic application, you can ignore the following warning that may appear:

      ```
      Referenced assembly 'iAnywhere.Data.UltraLite.resources' is a localized
      satellite assembly
      ```

   b. From the **Debug** menu, choose **Start Debugging**.

      This action deploys your application to the device or emulator, and starts it. The application is deployed to the emulator or device location: *\Program Files\VBApp* or *\Program Files\CSApp* depending on your project name.

      The deployment may take some time.

   c. Confirm that the application deploys to the emulator or your target device and the form (**Form1**) you have designed is displayed correctly.

d. Shutdown the emulator or the application on your target device.

Lesson 2: Create UltraLite database

The following procedure (performed on your desktop PC) creates an UltraLite database using Sybase Central.

See "Create a database with the Create Database Wizard" [*UltraLite - Database Management and Reference*].

To create a database

1. From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.

2. Use the UltraLite plug-in for Sybase Central to create a database in the same directory as your application. In general, the default database characteristics provided by Sybase Central are suitable. Note the following characteristics:
   - **Database file name** *VBApp.udb* or *CSApp.udb*, depending on your application type.
   - **Collation sequence** Use the default collation.
   - **Use case-sensitive string comparisons** This option should not be on.
   - **Table name** Type **Names**.
   - **Columns** Create columns in the **Names** table with the following attributes:

Column Name	Data Type (Size)	Nulls	Unique	Default value
ID	integer	No	Yes (primary key)	global autoincrement
Name	varchar(30)	No	No	None

   - **Primary key** Specify the ID column as primary key.

3. Exit Sybase Central and verify the database file is created in the required directory.

4. Link the initialized (empty) database file to your Visual Studio project so that the database file is deployed to the device along with the application code:
   - From the **Visual Studio** menu, choose **Project** » **Add Existing Item**.
   - Ensure that **Objects of Type** is set to **All Files**. Browse to the directory where you created the database file and select the file *VBApp.udb* or *CSApp.udb* depending on your application type.
   - Click the arrow in the **Add** button and click **Add As Link**.
   - In the Solution Explorer frame, right click the database file name that has just been added to the project and choose **Properties**.

     In the properties panel, set the **Build Action** property to **Content**; set the **Copy to Output Directory** property to **Copy always**.

Lesson 3: Connect to database

The following procedure adds a control to your UltraLite.NET application that establishes a connection to an UltraLite database.

To add an UltraLite connection to your application

1. Double-click the form to open the source file (*Form1.cs* or *Form1.vb*).

2. Add code to import the iAnywhere.Data.UltraLite namespace.

   Add the following statement as the very first line of the file.

   ```
   //Visual C#
   using iAnywhere.Data.UltraLite;

   'Visual Basic
   Imports iAnywhere.Data.UltraLite
   ```

3. Add global variables to the form declaration.

   For Visual C#, add the following code after the code describing the form components and before the first method declaration.

   ```
   //Visual C#
   private ULConnection Conn;
   private int[] ids;
   ```

   For a Visual Basic, add the following code at the beginning of the Form1 class.

   ```
   'Visual Basic
   Dim Conn As ULConnection
   Dim ids() As Integer
   ```

   These variables are used as follows:

   - **ULConnection** A Connection object is the root object for all actions executed on a connection to a database.
   - **ids** The ids array is used to hold the ID column values returned after executing a query.

     Although the ListBox control itself allows you access to sequential numbers, those numbers differ from the value of the ID column once a row has been deleted. For this reason, the ID column values must be stored separately.

4. Double-click a blank area of your form to create a Form1_Load method.

   This method performs the following tasks:

   - Open a connection to the database using the connection parameters set in the ulConnectionParms1 control.
   - Call the RefreshListBox method (defined later in this tutorial).

- Print (display) and error message if an error occurs. For SQL Anywhere errors, the code also prints the error code. See "Error Messages".

For C#, add the following code to the Form1_Load method.

```
//Visual C#
try {
    String ConnString = "dbf=\\Program Files\\CSApp\\CSApp.udb";
    Conn = new ULConnection( ConnString );
    Conn.Open();
    Conn.DatabaseID = 1;
    RefreshListBox();
}
catch ( System.Exception t ) {
    MessageBox.Show( "Exception: " + t.Message);
}
```

For Visual Basic, add the following code to the Form1_Load method.

```
'Visual Basic
Try
    Dim ConnString as String = "dbf=\Program Files\VBApp\VBApp.udb"
    Conn = New ULConnection( ConnString )
    Conn.Open()
    Conn.DatabaseID = 1
    RefreshListBox()
Catch
    MsgBox("Exception: " + err.Description)
End Try
```

5. Build the project.

   From the **Build** menu, choose **Build Solution**. At this stage, you may receive a single error reported; for example in C#: `error CS0103: The name 'RefreshListBox' does not exist in the class or namespace 'CSApp.Form1'` because RefreshListBox is not yet declared. The next lesson adds that function.

   If you get other errors, you must correct them before proceeding. Check for common errors, such as case inconsistencies in C#. For example, **UltraLite** and **ULConnection** must match case exactly. In Visual Basic it is crucial to include the **Imports iAnywhere.Data.UltraLite** statement as described in Lesson 3.

Lesson 4: Insert, update, and delete data

In this lesson you add code to your application to modify the data in your database. The following procedures use Dynamic SQL. The same techniques can be performed using the Table API.

See "Accessing and manipulating data with the Table API" on page 10.

The following procedure creates a supporting method to maintain the listbox. This method is required for data manipulation methods created in the remaining procedures.

To add code to maintain the listbox

1. Right-click the form and choose **View Code**.
2. Add a method of the Form1 class to update and populate the listbox. This method carries out the following tasks:
   - Clears the listbox.
   - Instantiates a ULCommand object and assigns it a SELECT query that returns data from the Names table in the database.
   - Executes the query, returning a result set as a ULDataReader.
   - Instantiates an integer array with length equal to the number of rows in the result set.
   - Populates the listbox with the names returned in the ULDataReader and populates the integer array with the ids returned in the ULDataReader.
   - Closes the ULDataReader.
   - If an error occurs, prints the error message. For SQL errors, the code also prints the error code.

     See "Error Messages".

   For C#, add the following code to your application as a method of the Form1 class.

```
//Visual C#
private void RefreshListBox(){
    try{
        long NumRows;
        int I = 0;
        lbNames.Items.Clear();
        using( ULCommand cmd = Conn.CreateCommand() ){
            cmd.CommandText = "SELECT ID, Name FROM Names";
            using( ULDataReader dr = cmd.ExecuteReader()){
                dr.MoveBeforeFirst();
                NumRows = dr.RowCount;
                ids = new int[ NumRows ];
                while (dr.MoveNext())
                {
                    lbNames.Items.Add(
                    dr.GetString(1));
                    ids[ I ] = dr.GetInt32(0);
                    I++;
                }
            }
            txtName.Text = " ";
        }
    }
    catch( Exception err ){
        MessageBox.Show(
        "Exception in RefreshListBox: " + err.Message );
    }
}
```

   For Visual Basic, add the following code to your application as a method of the Form1 class.

```
'Visual Basic
Private Sub RefreshListBox()
    Try
        Dim cmd As ULCommand = Conn.CreateCommand()
```



```
            Dim I As Integer = 0
            lbNames.Items.Clear()
            cmd.CommandText = "SELECT ID, Name FROM Names"
            Dim dr As ULDataReader = cmd.ExecuteReader()
            ReDim ids(dr.RowCount)
            While (dr.MoveNext)
                lbNames.Items.Add(dr.GetString(1))
                ids(I) = dr.GetInt32(0)
                I = I + 1
            End While
            dr.Close()
            txtName.Text = " "
        Catch ex As Exception
            MsgBox(ex.ToString)
        End Try
    End Sub
```

3. Build the project.

   Building the project should result in no errors.

To implement INSERT, UPDATE, and DELETE

1. On the form design tab, double-click **Insert** to create a btnInsert_Click method. This method carries out the following tasks:

   - Instantiates a ULCommand object and assigns it an INSERT statement that inserts the value in the text box into the database.
   - Executes the statement.
   - Disposes of the ULCommand object.
   - Refreshes the listbox.
   - If an error occurs, prints the error message. For SQL errors, the code also prints the error code.

     See "Error Messages".

   For C#, add the following code to the btnInsert_Click method.

   ```
   //Visual C#
   try {
       long RowsInserted;
       using( ULCommand cmd = Conn.CreateCommand() ) {
           cmd.CommandText =
               "INSERT INTO Names(name) VALUES (?)";
           cmd.Parameters.Add("", txtName.Text);
           RowsInserted = cmd.ExecuteNonQuery();
       }
       RefreshListBox();
   }
   catch( Exception err ) {
       MessageBox.Show("Exception: " + err.Message );
   }
   ```

   For Visual Basic, add the following code to the btnInsert_Click method.

   ```
   'Visual Basic
   Try
       Dim RowsInserted As Long
   ```

```
        Dim cmd As ULCommand = Conn.CreateCommand()
        cmd.CommandText = "INSERT INTO Names(name) VALUES (?)"
        cmd.Parameters.Add("", txtName.Text)
        RowsInserted = cmd.ExecuteNonQuery()
        cmd.Dispose()
        RefreshListBox()
    Catch
        MsgBox("Exception: " + Err.Description)
    End Try
```

2. On the form design tab, double-click **Update** to create a btnUpdate_Click method. This method carries out the following tasks:

   - Instantiates a ULCommand object and assigns it an UPDATE statement that inserts the value in the text box into the database based on the associated ID.
   - Executes the statement.
   - Disposes of the ULCommand object.
   - Refreshes the listbox.
   - If an error occurs, prints the error message. For SQL errors, the code also prints the error code.

     See "Error Messages".

   For C#, add the following code to the btnUpdate_Click method.

```
//Visual C#
try {
    long RowsUpdated;
    int updateID = ids[ lbNames.SelectedIndex ];
    using( ULCommand cmd = Conn.CreateCommand() ){
        cmd.CommandText =
            "UPDATE Names SET name = ? WHERE id = ?" ;
        cmd.Parameters.Add("", txtName.Text );
        cmd.Parameters.Add("", updateID);
        RowsUpdated = cmd.ExecuteNonQuery();
    }
    RefreshListBox();
}
catch( Exception err ) {
    MessageBox.Show(
        "Exception: " + err.Message);
}
```

   For Visual Basic, add the following code to the btnUpdate_Click method.

```
'Visual Basic
Try
    Dim RowsUpdated As Long
    Dim updateID As Integer = ids(lbNames.SelectedIndex)
    Dim cmd As ULCommand = Conn.CreateCommand()
    cmd.CommandText = "UPDATE Names SET name = ? WHERE id = ?"
    cmd.Parameters.Add("", txtName.Text)
    cmd.Parameters.Add("", updateID)
    RowsUpdated = cmd.ExecuteNonQuery()
    cmd.Dispose()
    RefreshListBox()
Catch
    MsgBox("Exception: " + Err.Description)
End Try
```


3. On the form design tab, double-click **Delete** to create a btnDelete_Click method. Add code to carry out the following tasks:

   - Instantiates a ULCommand object and assigns it a DELETE statement. The DELETE statement deletes the selected row from the database, based on the associated ID from the integer array ids.
   - Executes the statement.
   - Disposes of the ULCommand object.
   - Refreshes the listbox.
   - If an error occurs, displays the error message. For SQL errors, the code also displays the error code.

     See "Error Messages".

   For C#, add the following code to the btnDelete_Click method.

   ```
   //Visual C#
   try{
       long RowsDeleted;
       int deleteID = ids[lbNames.SelectedIndex];
       using( ULCommand cmd = Conn.CreateCommand() ){
           cmd.CommandText =
               "DELETE From Names WHERE id = ?" ;
           cmd.Parameters.Add("", deleteID);
           RowsDeleted = cmd.ExecuteNonQuery ();
       }
       RefreshListBox();
   }
   catch( Exception err ) {
       MessageBox.Show("Exception: " + err.Message );
   }
   ```

   For Visual Basic, add the following code to the btnDelete_Click method.

   ```
   'Visual Basic
   Try
       Dim RowsDeleted As Long
       Dim deleteID As Integer = ids(lbNames.SelectedIndex)
       Dim cmd As ULCommand = Conn.CreateCommand()
       cmd.CommandText = "DELETE From Names WHERE id = ?"
       cmd.Parameters.Add("", deleteID)
       RowsDeleted = cmd.ExecuteNonQuery()
       cmd.Dispose()
       RefreshListBox()
   Catch
       MsgBox("Exception: " + Err.Description)
   End Try
   ```

4. Build your application to confirm that it compiles properly.

Lesson 5: Build and deploy application

In the following procedure, you build your application and deploy it to a remote device or emulator.

To deploy your application

1. Build the solution.

   Ensure that your application builds without errors.

2. Choose the deployment target.

   The deployment target must match the version of *ulnet12.dll* that you included in your application.

3. Choose **Debug** » **Start**.

   This builds an executable file containing your application and deploys it to the emulator. The process may take some time, especially if it must deploy the .NET Compact Framework before running the application.

Deployment troubleshooting checklist

If errors are reported, you may want to check that your deployment was completed successfully using the following checklist:

- Confirm that the application is deployed into *\Program Files\appname*, where *appname* is the name you gave your application in Lesson 1 (CSApp or VBApp).
- Confirm that the path to the database file in your application code is correct. See "Lesson 3: Connect to database" on page 26.
- Confirm that you chose Link File when adding the database file to the project and you set the Build Action to Content Only and Copy to Output Directory is set to Copy Always. If you did not set these options correctly, the files will not be deployed to the device.
- Ensure that you added a reference to the correct version of *ulnet12.dll* for your target platform, or ran the Windows Mobile installer. For versions of Windows Mobile earlier than Windows Mobile 5.0, if you switch between the emulator and a real device, you must change the version of the library that you use. See "Lesson 1: Create Visual Studio project" on page 21.
- You may want to exit the emulator without saving the emulator state. Redeploying the application copies all required files to the emulator, and ensures there are no version problems.

To test your application

1. Insert data into the database:

   Enter a name in the text box and click **Insert**. The name should now appear in the listbox.

2. Update data in the database:

   Select a name from the listbox. Enter a new name in the text box. Click **Update**. The new name should now appear in place of the old name in the listbox.

3. Delete data from the database:

Select a name from the list. Click **Delete**. The name no longer appears in the list.

This tutorial is complete.

Code listing for C# tutorial

Following is the complete code for the tutorial program described in the preceding sections.

```
using iAnywhere.Data.UltraLite;
using System;
using System.Linq;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace CSApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private ULConnection Conn;
        private int[] ids;

        private void Form1_Load(object sender, EventArgs e)
        {
            try
            {
                String ConnString = "dbf=\\Program Files\\CSApp\\CSApp.udb";
                Conn = new ULConnection(ConnString);
                Conn.Open();
                Conn.DatabaseID = 1;
                RefreshListBox();
            }
            catch (System.Exception t)
            {
                MessageBox.Show("Exception: " + t.Message);
            }
        }
        private void RefreshListBox()
        {
            try
            {
                long NumRows;
                int I = 0;
                lbNames.Items.Clear();
                using (ULCommand cmd = Conn.CreateCommand())
                {
                    cmd.CommandText = "SELECT ID, Name FROM Names";
                    using (ULDataReader dr = cmd.ExecuteReader())
                    {
                        dr.MoveBeforeFirst();
                        NumRows = dr.RowCount;
                        ids = new int[NumRows];
```

```
                while (dr.MoveNext())
                {
                    lbNames.Items.Add(
                    dr.GetString(1));
                    ids[i] = dr.GetInt32(0);
                    I++;
                }
            }
            txtName.Text = " ";
        }
    }
    catch (Exception err)
    {
        MessageBox.Show(
        "Exception in RefreshListBox: " + err.Message);
    }
}

private void btnInsert_Click(object sender, EventArgs e)
{
    try
    {
        long RowsInserted;
        using (ULCommand cmd = Conn.CreateCommand())
        {
            cmd.CommandText =
                "INSERT INTO Names(name) VALUES (?)";
            cmd.Parameters.Add("", txtName.Text);
            RowsInserted = cmd.ExecuteNonQuery();
        }
        RefreshListBox();
    }
    catch (Exception err)
    {
        MessageBox.Show("Exception: " + err.Message);
    }
}

private void btnUpdate_Click(object sender, EventArgs e)
{
    try
    {
        long RowsUpdated;
        int updateID = ids[lbNames.SelectedIndex];
        using (ULCommand cmd = Conn.CreateCommand())
        {
            cmd.CommandText =
                "UPDATE Names SET name = ? WHERE id = ?";
            cmd.Parameters.Add("", txtName.Text);
            cmd.Parameters.Add("", updateID);
            RowsUpdated = cmd.ExecuteNonQuery();
        }
        RefreshListBox();
    }
    catch (Exception err)
    {
        MessageBox.Show(
            "Exception: " + err.Message);
    }
}

private void btnDelete_Click(object sender, EventArgs e)
{
    try
```



```
            {
                long RowsDeleted;
                int deleteID = ids[lbNames.SelectedIndex];
                using (ULCommand cmd = Conn.CreateCommand())
                {
                    cmd.CommandText =
                        "DELETE From Names WHERE id = ?";
                    cmd.Parameters.Add("", deleteID);
                    RowsDeleted = cmd.ExecuteNonQuery();
                }
                RefreshListBox();
            }
            catch (Exception err)
            {
                MessageBox.Show("Exception: " + err.Message);
            }
        }
    }
}
```

Code listing for Visual Basic tutorial

```
Imports iAnywhere.Data.UltraLite
Public Class Form1
    Dim Conn As ULConnection
    Dim ids() As Integer
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles MyBase.Load
        Try
            Dim ConnString As String = "dbf=\Program Files\VBApp\VBApp.udb"
            Conn = New ULConnection(ConnString)
            Conn.Open()
            Conn.DatabaseID = 1
            RefreshListBox()
        Catch
            MsgBox("Exception: " + Err.Description)
        End Try
    End Sub
    Private Sub RefreshListBox()
        Try
            Dim cmd As ULCommand = Conn.CreateCommand()
            Dim I As Integer = 0
            lbNames.Items.Clear()
            cmd.CommandText = "SELECT ID, Name FROM Names"
            Dim dr As ULDataReader = cmd.ExecuteReader()
            ReDim ids(dr.RowCount)
            While (dr.MoveNext)
                lbNames.Items.Add(dr.GetString(1))
                ids(I) = dr.GetInt32(0)
                I = I + 1
            End While
            dr.Close()
            txtName.Text = " "
        Catch ex As Exception
            MsgBox(ex.ToString)
        End Try
    End Sub

    Private Sub btnInsert_Click(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles btnInsert.Click
```

```
            Try
                Dim RowsInserted As Long
                Dim cmd As ULCommand = Conn.CreateCommand()
                cmd.CommandText = "INSERT INTO Names(name) VALUES (?)"
                cmd.Parameters.Add("", txtName.Text)
                RowsInserted = cmd.ExecuteNonQuery()
                cmd.Dispose()
                RefreshListBox()
            Catch
                MsgBox("Exception: " + Err.Description)
            End Try
        End Sub

        Private Sub btnUpdate_Click(ByVal sender As System.Object, ByVal e As
          System.EventArgs) Handles btnUpdate.Click
            Try
                Dim RowsUpdated As Long
                Dim updateID As Integer = ids(lbNames.SelectedIndex)
                Dim cmd As ULCommand = Conn.CreateCommand()
                cmd.CommandText = "UPDATE Names SET name = ? WHERE id = ?"
                cmd.Parameters.Add("", txtName.Text)
                cmd.Parameters.Add("", updateID)
                RowsUpdated = cmd.ExecuteNonQuery()
                cmd.Dispose()
                RefreshListBox()
            Catch
                MsgBox("Exception: " + Err.Description)
            End Try
        End Sub

        Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal e As
          System.EventArgs) Handles btnDelete.Click
            Try
                Dim RowsDeleted As Long
                Dim deleteID As Integer = ids(lbNames.SelectedIndex)
                Dim cmd As ULCommand = Conn.CreateCommand()
                cmd.CommandText = "DELETE From Names WHERE id = ?"
                cmd.Parameters.Add("", deleteID)
                RowsDeleted = cmd.ExecuteNonQuery()
                cmd.Dispose()
                RefreshListBox()
            Catch
                MsgBox("Exception: " + Err.Description)
            End Try
        End Sub
    End Class
```


UltraLite.NET API reference

Namespace

```
iAnywhere.Data.UltraLite
```

ULActiveSyncListener interface

UL Ext: The listener interface for receiving ActiveSync events.

Visual Basic syntax

```
Public Interface ULActiveSyncListener
```

C# syntax

```
public interface ULActiveSyncListener
```

Members

All members of ULActiveSyncListener interface, including all inherited members.

Name	Description
"ActiveSyncInvoked method"	Invoked when the MobiLink provider for ActiveSync calls the application to perform synchronization.

ActiveSyncInvoked method

Invoked when the MobiLink provider for ActiveSync calls the application to perform synchronization.

Visual Basic syntax

```
Public Sub ActiveSyncInvoked(ByVal launchedByProvider As Boolean)
```

C# syntax

```
public void ActiveSyncInvoked(bool launchedByProvider)
```

Parameters

- **launchedByProvider** True if the application was launched by the MobiLink provider to perform ActiveSync synchronization. The application must then shut itself down after it has finished synchronizing. False if the application was already running when called by the MobiLink provider for ActiveSync.

Remarks

This method is invoked by a separate thread. To avoid multi-threading issues, it should post an event to the UI. If you are using multi-threading, it is recommended that you use a separate connection and use the lock keyword to access any objects shared with the rest of the application.

Once synchronization has completed, applications should call ULDatabaseManager.SignalSyncIsComplete() to signal the MobiLink provider for ActiveSync.

See also

- "SignalSyncIsComplete method" on page 211

Example

```
' Visual Basic
Imports iAnywhere.Data.UltraLite
Public Class MainWindow
    Inherits System.Windows.Forms.Form
    Implements ULActiveSyncListener

    Private conn As ULConnection

    Public Sub New(ByVal args() As String)
        MyBase.New()

        ' This call is required by the Windows Form Designer.
        InitializeComponent()

        ' Add any initialization after the InitializeComponent call.
        ULConnection.DatabaseManager.SetActiveSyncListener( _
            "myCompany.myapp", Me _
        )

        ' Create Connection
        ...
    End Sub

    Protected Overrides Sub OnClosing( _
        ByVal e As System.ComponentModel.CancelEventArgs _
    )
        ULConnection.DatabaseManager.SetActiveSyncListener( _
            Nothing, Nothing _
        )
        MyBase.OnClosing(e)
    End Sub

    Public Sub ActiveSyncInvoked( _
        ByVal launchedByProvider As Boolean _
    ) Implements ULActiveSyncListener.ActiveSyncInvoked
        Me.Invoke(New EventHandler(AddressOf Me.ActiveSyncAction))
    End Sub

    Public Sub ActiveSyncAction( _
        ByVal sender As Object, ByVal e As EventArgs _
    )
        ' Perform active sync.
        conn.Synchronize()
        ULConnection.DatabaseManager.SignalSyncIsComplete()
    End Sub
End Class
```


The following code is the C# language equivalent:

```
// C#
using iAnywhere.Data.UltraLite;

public class Form1 : System.Windows.Forms.Form, ULActiveSyncListener
{
    private System.Windows.Forms.MainMenu mainMenu1;
    private ULConnection conn;

    public Form1()
    {
        //
        // Required for Windows Form Designer support.
        // InitializeComponent();

        //
        // TODO: Add any constructor code after the
        // InitializeComponent call.
        //
        ULConnection.DatabaseManager.SetActiveSyncListener( _
            "myCompany.myapp", this _
        );

        // Create connection
        ...
    }

    protected override void Dispose( bool disposing )
    {
        base.Dispose( disposing );
    }

    protected override void OnClosing( _
        System.ComponentModel.CancelEventArgs e )
    {
        ULConnection.DatabaseManager.SetActiveSyncListener(null, null);
        base.OnClosing(e);
    }

    public void ActiveSyncInvoked(bool launchedByProvider)
    {
        this.Invoke( new EventHandler( ActiveSyncHandler ) );
    }

    internal void ActiveSyncHandler(object sender, EventArgs e)
    {
        conn.Synchronize();
        ULConnection.DatabaseManager.SignalSyncIsComplete();
    }
}
```

ULBulkCopy class

Efficiently bulk load an UltraLite table with data from another source.

Visual Basic syntax

```
Public NotInheritable Class ULBulkCopy Implements System.IDisposable
```

C# syntax

```
public sealed class ULBulkCopy : System.IDisposable
```

Base classes

- System.IDisposable

Members

All members of ULBulkCopy class, including all inherited members.

Name	Description
"ULBulkCopy constructor"	Initializes a ULBulkCopy object with the specified ULConnection.
"Close method"	Closes the ULBulkCopy instance.
"Dispose method"	Disposes of the ULBulkCopy instance.
"WriteToServer method"	Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName field of the ULBulkCopy object.
"BatchSize property"	Gets or sets the number of rows in each batch.
"BulkCopyTimeout property"	Gets or sets the number of seconds for the operation to complete before it times out.
"ColumnMappings property"	Returns a collection of ULBulkCopyColumnMapping items.
"DestinationTableName property"	Gets or sets the name of the destination table on the server.
"NotifyAfter property"	Specifies the number of rows to be processed before generating a notification event.
"ULRowsCopied event"	This event occurs every time the number of rows specified by NotifyAfter have been processed.

Remarks

The ULBulkCopy class is not available in the .NET Compact Framework 2.0.

ULBulkCopy constructor

Initializes a ULBulkCopy object with the specified ULConnection.

Overload list

Name	Description
"ULBulkCopy(string) constructor"	Initializes a ULBulkCopy object with the specified connection string.
"ULBulkCopy(string, ULBulkCopyOptions) constructor"	Initializes a ULBulkCopy object with the specified connection string and copy options.
"ULBulkCopy(ULConnection) constructor"	Initializes a ULBulkCopy object with the specified ULConnection.
"ULBulkCopy(ULConnection, ULBulkCopyOptions, ULTransaction) constructor"	Initializes a ULBulkCopy object with the specified ULConnection, copy options and ULTransaction.

ULBulkCopy(string) constructor

Initializes a ULBulkCopy object with the specified connection string.

Visual Basic syntax

```
Public Sub New(ByVal connectionString As String)
```

C# syntax

```
public ULBulkCopy(string connectionString)
```

Parameters

- **connectionString** The string defining the connection to be opened for use by the ULBulkCopy instance. A connection string is a semicolon-separated list of keyword=value pairs.

Remarks

The ULBulkCopy class is not available in the .NET Compact Framework 2.0.

This syntax opens a connection during WriteToServer using connectionString. The connection is closed at the end of WriteToServer.

The connection string can be supplied using a ULConnectionParms object.

See also

- "ConnectionString property" on page 145
- "ULConnectionParms class" on page 154
- System.IDisposable

ULBulkCopy(string, ULBulkCopyOptions) constructor

Initializes a ULBulkCopy object with the specified connection string and copy options.

Visual Basic syntax

```
Public Sub New(
    ByVal connectionString As String,
    ByVal copyOptions As ULBulkCopyOptions
)
```

C# syntax

```
public ULBulkCopy(
    string connectionString,
    ULBulkCopyOptions copyOptions
)
```

Parameters

- **connectionString** The string defining the connection to be opened for use by the ULBulkCopy instance. A connection string is a semicolon-separated list of keyword=value pairs.
- **copyOptions** A combination of values from the ULBulkCopyOptions enumeration that determines how data source rows are copied to the destination table.

Remarks

The ULBulkCopy class is not available in the .NET Compact Framework 2.0.

This syntax opens a connection during WriteToServer using connectionString. The connection is closed at the end of WriteToServer.

See also

- "ConnectionString property" on page 145
- "ULBulkCopyOptions enumeration" on page 436

ULBulkCopy(ULConnection) constructor

Initializes a ULBulkCopy object with the specified ULConnection.

Visual Basic syntax

```
Public Sub New(ByVal connection As ULConnection)
```

C# syntax

```
public ULBulkCopy(ULConnection connection)
```

Parameters

- **connection** The already open ULConnection that is used to perform the bulk-copy operation. If the connection is not open, an exception is thrown in WriteToServer.

Remarks

The ULBulkCopy class is not available in the .NET Compact Framework 2.0.

See also

- "ULConnection class" on page 110

ULBulkCopy(ULConnection, ULBulkCopyOptions, ULTransaction) constructor

Initializes a ULBulkCopy object with the specified ULConnection, copy options and ULTransaction.

Visual Basic syntax

```
Public Sub New(
    ByVal connection As ULConnection,
    ByVal copyOptions As ULBulkCopyOptions,
    ByVal externalTransaction As ULTransaction
)
```

C# syntax

```
public ULBulkCopy(
    ULConnection connection,
    ULBulkCopyOptions copyOptions,
    ULTransaction externalTransaction
)
```

Parameters

- **connection** The already open ULConnection that is used to perform the bulk-copy operation. If the connection is not open, an exception is thrown in WriteToServer.
- **copyOptions** A combination of values from the ULBulkCopyOptions enumeration that determines how data source rows are copied to the destination table.
- **externalTransaction** An existing ULTransaction instance under which the bulk copy occurs. If externalTransaction is not a null reference (Nothing in Visual Basic), then the bulk-copy operation is done within it. It is an error to specify both an external transaction and the ULBulkCopyOptions.UseInternalTransaction option.

Remarks

The ULBulkCopy class is not available in the .NET Compact Framework 2.0.

See also

- "ULConnection class" on page 110
- "ULTransaction class" on page 430

Close method

Closes the ULBulkCopy instance.

Visual Basic syntax

```
Public Sub Close()
```

C# syntax

```
public void Close()
```

Dispose method

Disposes of the ULBulkCopy instance.

Visual Basic syntax

```
Public Sub Dispose()
```

C# syntax

```
public void Dispose()
```

WriteToServer method

Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName field of the ULBulkCopy object.

Overload list

Name	Description
"WriteToServer(DataRow[]) method"	Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName field of the ULBulkCopy object.
"WriteToServer(DataTable) method"	Copies all rows in the supplied System.Data.DataTable to a destination table specified by the DestinationTableName of the ULBulkCopy object.
"WriteToServer(DataTable, DataRowState) method"	Copies all rows in the supplied System.Data.DataTable with the specified row state to a destination table specified by the DestinationTableName of the ULBulkCopy object.
"WriteToServer(IDataReader) method"	Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the DestinationTableName of the ULBulkCopy object.

WriteToServer(DataRow[]) method

Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName field of the ULBulkCopy object.

Visual Basic syntax

```
Public Sub WriteToServer(ByVal rows As DataRow())
```

C# syntax

```
public void WriteToServer(DataRow[] rows)
```

Parameters

- **rows** An array of System.Data.DataRow objects to be copied to the destination table.

See also

- "DestinationTableName property" on page 48
- System.Data.DataRow

WriteToServer(DataTable) method

Copies all rows in the supplied System.Data.DataTable to a destination table specified by the DestinationTableName of the ULBulkCopy object.

Visual Basic syntax

```
Public Sub WriteToServer(ByVal table As DataTable)
```

C# syntax

```
public void WriteToServer(DataTable table)
```

Parameters

- **table** A System.Data.DataTable whose rows to be copied to the destination table.

See also

- "DestinationTableName property" on page 48
- System.Data.DataTable

WriteToServer(DataTable, DataRowState) method

Copies all rows in the supplied System.Data.DataTable with the specified row state to a destination table specified by the DestinationTableName of the ULBulkCopy object.

Visual Basic syntax

```
Public Sub WriteToServer(
    ByVal table As DataTable,
    ByVal rowState As DataRowState
)
```

C# syntax

```
public void WriteToServer(DataTable table, DataRowState rowState)
```

Parameters

- **table** A System.Data.DataTable whose rows to be copied to the destination table.
- **rowState** A value from the System.Data.DataRowState enumeration. Only rows matching the row state are copied to the destination.

Remarks

If rowState is specified, then only those rows that have the same row state are copied.

See also

- "DestinationTableName property" on page 48
- System.Data.DataTable

WriteToServer(IDataReader) method

Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the DestinationTableName of the ULBulkCopy object.

Visual Basic syntax

```
Public Sub WriteToServer(ByVal reader As IDataReader)
```

C# syntax

```
public void WriteToServer(IDataReader reader)
```

Parameters

- **reader** A System.Data.IDataReader whose rows to be copied to the destination table.

See also

- "DestinationTableName property" on page 48
- System.Data.IDataReader

BatchSize property

Gets or sets the number of rows in each batch.

Visual Basic syntax

```
Public Property BatchSize As Integer
```

C# syntax

```
public int BatchSize {get;set;}
```


Remarks

At the end of each batch, the rows in the batch are sent to the server.

The number of rows in each batch. The default is 0.

Setting it to zero causes all the rows to be sent in one batch.

Setting it less than zero is an error.

If this value is changed while a batch is in progress, the current batch completes and any further batches use the new value.

BulkCopyTimeout property

Gets or sets the number of seconds for the operation to complete before it times out.

Visual Basic syntax

```
Public Property BulkCopyTimeout As Integer
```

C# syntax

```
public int BulkCopyTimeout {get;set;}
```

Remarks

The default value is 30 seconds.

A value of zero indicates no limit. This should be avoided because it may cause an indefinite wait.

If the operation times out, then all rows in the current transaction are rolled back and an SAException is raised.

Setting it less than zero is an error.

ColumnMappings property

Returns a collection of ULBulkCopyColumnMapping items.

Visual Basic syntax

```
Public ReadOnly Property ColumnMappings As
ULBulkCopyColumnMappingCollection
```

C# syntax

```
public ULBulkCopyColumnMappingCollection ColumnMappings {get;}
```

Remarks

Column mappings define the relationships between columns in the data source and columns in the destination.

By default, it is an empty collection.

The property cannot be modified while WriteToServer is executing.

If ColumnMappings is empty when WriteToServer is executed, then the first column in the source is mapped to the first column in the destination, the second to the second, and so on. This takes place as long as the column types are convertible, there are at least as many destination columns as source columns, and any extra destination columns are nullable.

See also

- "ULBulkCopyColumnMapping class" on page 49

DestinationTableName property

Gets or sets the name of the destination table on the server.

Visual Basic syntax

```
Public Property DestinationTableName As String
```

C# syntax

```
public string DestinationTableName {get;set;}
```

Remarks

The default value is a null reference (Nothing in Visual Basic).

If the value is changed while WriteToServer is executing, the change has no effect.

If the value has not been set before a call to WriteToServer, an InvalidOperationException is raised.

It is an error to set the value to null (Nothing in Visual Basic) or the empty string.

NotifyAfter property

Specifies the number of rows to be processed before generating a notification event.

Visual Basic syntax

```
Public Property NotifyAfter As Integer
```

C# syntax

```
public int NotifyAfter {get;set;}
```


Remarks

An integer representing the number of rows to be processed before generating a notification event, or zero is if the property has not been set.

Changes made to NotifyAfter, while executing WriteToServer, do not take effect until after the next notification.

Setting it less than zero is an error.

The value of NotifyAfter and BulkCopyTimeOut are mutually exclusive, so the event can fire even if no rows have been sent to the database or committed.

See also

- "BulkCopyTimeout property" on page 47

ULRowsCopied event

This event occurs every time the number of rows specified by NotifyAfter have been processed.

Visual Basic syntax

```
Public Event ULRowsCopied As ULRowsCopiedEventHandler
```

C# syntax

```
public event ULRowsCopiedEventHandler ULRowsCopied;
```

Remarks

The receipt of a ULRowsCopied event does not imply that any rows have been committed. You cannot call the Close method from this event.

See also

- "NotifyAfter property" on page 48

ULBulkCopyColumnMapping class

Defines the mapping between a column in a ULBulkCopy instance's data source and a column in the instance's destination table.

Visual Basic syntax

```
Public NotInheritable Class ULBulkCopyColumnMapping
```

C# syntax

```
public sealed class ULBulkCopyColumnMapping
```

Members

All members of ULBulkCopyColumnMapping class, including all inherited members.

Name	Description
"ULBulkCopyColumnMapping constructor"	Creates a new column mapping.
"DestinationColumn property"	Specifies the name of the column in the destination database table being mapped to.
"DestinationOrdinal property"	Specifies the ordinal value of the column in the destination database table being mapped to.
"SourceColumn property"	Specifies the name of the column being mapped in the data source.
"SourceOrdinal property"	Specifies the ordinal position of the source column within the data source.

Remarks

The ULBulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

See also

- "ULBulkCopy class" on page 39

ULBulkCopyColumnMapping constructor

Creates a new column mapping.

Overload list

Name	Description
"ULBulkCopyColumnMapping() constructor"	Creates a new column mapping.
"ULBulkCopyColumnMapping(int, int) constructor"	Creates a new column mapping, using column ordinals or names to refer to source and destination columns.
"ULBulkCopyColumnMapping(int, string) constructor"	Creates a new column mapping, using a column ordinal to refer to the source column and a column name to refer to the destination column.
"ULBulkCopyColumnMapping(string, int) constructor"	Creates a new column mapping, using a column name to refer to the source column and a column ordinal to refer to the destination the column.

Name	Description
"ULBulkCopyColumnMapping(string, string) constructor"	Creates a new column mapping, using column names to refer to source and destination columns.

ULBulkCopyColumnMapping() constructor

Creates a new column mapping.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ULBulkCopyColumnMapping()
```

Remarks

The ULBulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

ULBulkCopyColumnMapping(int, int) constructor

Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

Visual Basic syntax

```
Public Sub New(
    ByVal sourceColumnOrdinal As Integer,
    ByVal destinationColumnOrdinal As Integer
)
```

C# syntax

```
public ULBulkCopyColumnMapping(
    int sourceColumnOrdinal,
    int destinationColumnOrdinal
)
```

Parameters

- **sourceColumnOrdinal** The ordinal position of the source column within the data source. The first column in a data source has ordinal position zero.
- **destinationColumnOrdinal** The ordinal position of the destination column within the destination table. The first column in a table has ordinal position zero.

Remarks

The ULBulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

ULBulkCopyColumnMapping(int, string) constructor

Creates a new column mapping, using a column ordinal to refer to the source column and a column name to refer to the destination column.

Visual Basic syntax

```
Public Sub New(
    ByVal sourceColumnOrdinal As Integer,
    ByVal destinationColumn As String
)
```

C# syntax

```
public ULBulkCopyColumnMapping(
    int sourceColumnOrdinal,
    string destinationColumn
)
```

Parameters

- **sourceColumnOrdinal** The ordinal position of the source column within the data source. The first column in a data source has ordinal position zero.
- **destinationColumn** The name of the destination column within the destination table.

Remarks

The ULBulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

ULBulkCopyColumnMapping(string, int) constructor

Creates a new column mapping, using a column name to refer to the source column and a column ordinal to refer to the destination the column.

Visual Basic syntax

```
Public Sub New(
    ByVal sourceColumn As String,
    ByVal destinationColumnOrdinal As Integer
)
```

C# syntax

```
public ULBulkCopyColumnMapping(
    string sourceColumn,
    int destinationColumnOrdinal
)
```

Parameters

- **sourceColumn** The name of the source column within the data source.
- **destinationColumnOrdinal** The ordinal position of the destination column within the destination table. The first column in a table has ordinal position zero.

Remarks

The ULBulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

ULBulkCopyColumnMapping(string, string) constructor

Creates a new column mapping, using column names to refer to source and destination columns.

Visual Basic syntax

```
Public Sub New(
    ByVal sourceColumn As String,
    ByVal destinationColumn As String
)
```

C# syntax

```
public ULBulkCopyColumnMapping(
    string sourceColumn,
    string destinationColumn
)
```

Parameters

- **sourceColumn** The name of the source column within the data source.
- **destinationColumn** The name of the destination column within the destination table.

Remarks

The ULBulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

DestinationColumn property

Specifies the name of the column in the destination database table being mapped to.

Visual Basic syntax

```
Public Property DestinationColumn As String
```

C# syntax

```
public string DestinationColumn {get;set;}
```

Remarks

A string specifying the name of the column in the destination table or a null reference (Nothing in Visual Basic) if the DestinationOrdinal has priority.

The DestinationColumn and DestinationOrdinal properties are mutually exclusive. The most recently set value takes priority.

Setting the DestinationColumn property causes the DestinationOrdinal property to be set to -1. Setting the DestinationOrdinal property causes the DestinationColumn property to be set to a null reference (Nothing in Visual Basic).

It is an error to set DestinationColumn to null or the empty string.

See also

- "DestinationOrdinal property" on page 54

DestinationOrdinal property

Specifies the ordinal value of the column in the destination database table being mapped to.

Visual Basic syntax

```
Public Property DestinationOrdinal As Integer
```

C# syntax

```
public int DestinationOrdinal {get;set;}
```

Remarks

An integer specifying the ordinal of the column being mapped to in the destination table or -1 if the property is not set.

The DestinationColumn and DestinationOrdinal properties are mutually exclusive. The most recently set value takes priority.

Setting the DestinationColumn property causes the DestinationOrdinal property to be set to -1. Setting the DestinationOrdinal property causes the DestinationColumn property to be set to a null reference (Nothing in Visual Basic).

See also

- "DestinationColumn property" on page 53

SourceColumn property

Specifies the name of the column being mapped in the data source.

Visual Basic syntax

```
Public Property SourceColumn As String
```

C# syntax

```
public string SourceColumn {get;set;}
```


Remarks

A string specifying the name of the column in the data source or a null reference (Nothing in Visual Basic) if the SourceOrdinal has priority.

The SourceColumn and SourceOrdinal properties are mutually exclusive. The most recently set value takes priority.

Setting the SourceColumn property causes the SourceOrdinal property to be set to -1. Setting the SourceOrdinal property causes the SourceColumn property to be set to a null reference (Nothing in Visual Basic).

It is an error to set SourceColumn to null or the empty string.

See also

- "SourceOrdinal property" on page 55

SourceOrdinal property

Specifies the ordinal position of the source column within the data source.

Visual Basic syntax

```
Public Property SourceOrdinal As Integer
```

C# syntax

```
public int SourceOrdinal {get;set;}
```

Remarks

An integer specifying the ordinal of the column in the data source or -1 if the property is not set.

The SourceColumn and SourceOrdinal properties are mutually exclusive. The most recently set value takes priority.

Setting the SourceColumn property causes the SourceOrdinal property to be set to -1. Setting the SourceOrdinal property causes the SourceColumn property to be set to a null reference (Nothing in Visual Basic).

See also

- "SourceColumn property" on page 54

ULBulkCopyColumnMappingCollection class

A collection of ULBulkCopyColumnMapping objects that inherits from System.Collections.CollectionBase.

Visual Basic syntax

```
Public NotInheritable Class ULBulkCopyColumnMappingCollection
    Inherits System.Collections.CollectionBase
```

C# syntax

```
public sealed class ULBulkCopyColumnMappingCollection :
    System.Collections.CollectionBase
```

Base classes

- System.Collections.CollectionBase

Members

All members of ULBulkCopyColumnMappingCollection class, including all inherited members.

Name	Description
"Add method"	Adds the specified ULBulkCopyColumnMapping to the collection.
Clear method (Inherited from System.Collections.CollectionBase)	Removes all objects from the System.Collections.CollectionBase instance.
"Contains method"	Returns whether the specified ULBulkCopyColumnMapping object exists in the collection.
"CopyTo method"	Copies the elements of the ULBulkCopyColumnMappingCollection to an array of ULBulkCopyColumnMapping items, starting at a particular index.
GetEnumerator method (Inherited from System.Collections.CollectionBase)	Returns an enumerator that iterates through the System.Collections.CollectionBase instance.
"IndexOf method"	Returns the index of the specified ULBulkCopyColumnMapping within the collection.
OnClear method (Inherited from System.Collections.CollectionBase)	Performs additional custom processes when clearing the contents of the System.Collections.CollectionBase instance.
OnClearComplete method (Inherited from System.Collections.CollectionBase)	Performs additional custom processes after clearing the contents of the System.Collections.CollectionBase instance.
OnInsert method (Inherited from System.Collections.CollectionBase)	Performs additional custom processes before inserting a new element into the System.Collections.CollectionBase instance.
OnInsertComplete method (Inherited from System.Collections.CollectionBase)	Performs additional custom processes after inserting a new element into the System.Collections.CollectionBase instance.

Name	Description
OnRemove method (Inherited from System.Collections.CollectionBase)	Performs additional custom processes when removing an element from the System.Collections.CollectionBase instance.
OnRemoveComplete method (Inherited from System.Collections.CollectionBase)	Performs additional custom processes after removing an element from the System.Collections.CollectionBase instance.
OnSet method (Inherited from System.Collections.CollectionBase)	Performs additional custom processes before setting a value in the System.Collections.CollectionBase instance.
OnSetComplete method (Inherited from System.Collections.CollectionBase)	Performs additional custom processes after setting a value in the System.Collections.CollectionBase instance.
OnValidate method (Inherited from System.Collections.CollectionBase)	Performs additional custom processes when validating a value.
"Remove method"	Removes the specified ULBulkCopyColumnMapping element from the ULBulkCopyColumnMappingCollection.
"RemoveAt method"	Removes the mapping at the specified index from the collection.
Capacity property (Inherited from System.Collections.CollectionBase)	Gets or sets the number of elements that the System.Collections.CollectionBase can contain.
Count property (Inherited from System.Collections.CollectionBase)	Gets the number of elements contained in the System.Collections.CollectionBase instance.
InnerList property (Inherited from System.Collections.CollectionBase)	Gets an System.Collections.ArrayList containing the list of elements in the System.Collections.CollectionBase instance.
List property (Inherited from System.Collections.CollectionBase)	Gets an System.Collections.IList containing the list of elements in the System.Collections.CollectionBase instance.
"this property"	Gets the ULBulkCopyColumnMapping object at the specified index.

Remarks

The ULBulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

See also

- "ULBulkCopyColumnMapping class" on page 49

Add method

Adds the specified ULBulkCopyColumnMapping to the collection.

Overload list

Name	Description
"Add(int, int) method"	Creates a new ULBulkCopyColumnMapping instance using ordinals to specify both source and destination columns, and adds the mapping to the collection.
"Add(int, string) method"	Creates a new ULBulkCopyColumnMapping using a column ordinal to refer to the source column and a column name to refer to the destination column, and adds mapping to the collection.
"Add(string, int) method"	Creates a new ULBulkCopyColumnMapping using a column name to refer to the source column and a column ordinal to refer to the destination the column, and adds the mapping to the collection.
"Add(string, string) method"	Creates a new ULBulkCopyColumnMapping using column names to specify both source and destination columns, and adds the mapping to the collection.
"Add(ULBulkCopyColumnMapping) method"	Adds the specified ULBulkCopyColumnMapping to the collection.

Add(int, int) method

Creates a new ULBulkCopyColumnMapping instance using ordinals to specify both source and destination columns, and adds the mapping to the collection.

Visual Basic syntax

```
Public Function Add(
    ByVal sourceColumnOrdinal As Integer,
    ByVal destinationColumnOrdinal As Integer
) As ULBulkCopyColumnMapping
```

C# syntax

```
public ULBulkCopyColumnMapping Add(
    int sourceColumnOrdinal,
    int destinationColumnOrdinal
)
```

Parameters

- **sourceColumnOrdinal** The ordinal position of the source column within the data source. The first column in a data source has ordinal position zero.

- **destinationColumnOrdinal** The ordinal position of the destination column within the destination table. The first column in a table has ordinal position zero.

Remarks

The ULBulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

See also

- "ULBulkCopyColumnMapping class" on page 49

Add(int, string) method

Creates a new ULBulkCopyColumnMapping using a column ordinal to refer to the source column and a column name to refer to the destination column, and adds mapping to the collection.

Visual Basic syntax

```
Public Function Add(
    ByVal sourceColumnOrdinal As Integer,
    ByVal destinationColumn As String
) As ULBulkCopyColumnMapping
```

C# syntax

```
public ULBulkCopyColumnMapping Add(
    int sourceColumnOrdinal,
    string destinationColumn
)
```

Parameters

- **sourceColumnOrdinal** The ordinal position of the source column within the data source. The first column in a data source has ordinal position zero.
- **destinationColumn** The name of the destination column within the destination table.

Remarks

The ULBulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

See also

- "ULBulkCopyColumnMapping class" on page 49

Add(string, int) method

Creates a new ULBulkCopyColumnMapping using a column name to refer to the source column and a column ordinal to refer to the destination the column, and adds the mapping to the collection.

Visual Basic syntax

```
Public Function Add(
    ByVal sourceColumn As String,
```

```
    ByVal destinationColumnOrdinal As Integer
) As ULBulkCopyColumnMapping
```

C# syntax

```
public ULBulkCopyColumnMapping Add(
    string sourceColumn,
    int destinationColumnOrdinal
)
```

Parameters

- **sourceColumn** The name of the source column within the data source.
- **destinationColumnOrdinal** The ordinal position of the destination column within the destination table. The first column in a table has ordinal position zero.

Remarks

Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

The ULBulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

See also

- "ULBulkCopyColumnMapping class" on page 49

Add(string, string) method

Creates a new ULBulkCopyColumnMapping using column names to specify both source and destination columns, and adds the mapping to the collection.

Visual Basic syntax

```
Public Function Add(
    ByVal sourceColumn As String,
    ByVal destinationColumn As String
) As ULBulkCopyColumnMapping
```

C# syntax

```
public ULBulkCopyColumnMapping Add(
    string sourceColumn,
    string destinationColumn
)
```

Parameters

- **sourceColumn** The name of the source column within the data source.
- **destinationColumn** The name of the destination column within the destination table.

Remarks

The ULBulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

See also

- "ULBulkCopyColumnMapping class" on page 49

Add(ULBulkCopyColumnMapping) method

Adds the specified ULBulkCopyColumnMapping to the collection.

Visual Basic syntax

```
Public Function Add(
    ByVal bulkCopyColumnMapping As ULBulkCopyColumnMapping
) As ULBulkCopyColumnMapping
```

C# syntax

```
public ULBulkCopyColumnMapping Add(
    ULBulkCopyColumnMapping bulkCopyColumnMapping
)
```

Parameters

- **bulkCopyColumnMapping** The ULBulkCopyColumnMapping object that describes the mapping to be added to the collection.

Remarks

The ULBulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

See also

- "ULBulkCopyColumnMapping class" on page 49

Contains method

Returns whether the specified ULBulkCopyColumnMapping object exists in the collection.

Visual Basic syntax

```
Public Function Contains(
    ByVal value As ULBulkCopyColumnMapping
) As Boolean
```

C# syntax

```
public bool Contains(ULBulkCopyColumnMapping value)
```

Parameters

- **value** A valid ULBulkCopyColumnMapping object.

Returns

True if the specified mapping exists in the collection; otherwise, false.

See also

- "ULBulkCopyColumnMapping class" on page 49

CopyTo method

Copies the elements of the ULBulkCopyColumnMappingCollection to an array of ULBulkCopyColumnMapping items, starting at a particular index.

Visual Basic syntax

```
Public Sub CopyTo(
    ByVal array As ULBulkCopyColumnMapping(),
    ByVal index As Integer
)
```

C# syntax

```
public void CopyTo(ULBulkCopyColumnMapping[] array, int index)
```

Parameters

- **array** The one-dimensional ULBulkCopyColumnMapping array that is the destination of the elements copied from this ULBulkCopyColumnMappingCollection. The array must have zero-based indexing.
- **index** The zero-based index in the array at which copying begins.

See also

- "ULBulkCopyColumnMapping class" on page 49

IndexOf method

Returns the index of the specified ULBulkCopyColumnMapping within the collection.

Visual Basic syntax

```
Public Function IndexOf(
    ByVal value As ULBulkCopyColumnMapping
) As Integer
```

C# syntax

```
public int IndexOf(ULBulkCopyColumnMapping value)
```

Parameters

- **value** The ULBulkCopyColumnMapping object to search for.

Returns

The zero-based index of the column mapping is returned, or -1 is returned if the column mapping is not found in the collection.

See also

- "ULBulkCopyColumnMapping class" on page 49

Remove method

Removes the specified ULBulkCopyColumnMapping element from the ULBulkCopyColumnMappingCollection.

Visual Basic syntax

```
Public Sub Remove(ByVal value As ULBulkCopyColumnMapping)
```

C# syntax

```
public void Remove(ULBulkCopyColumnMapping value)
```

Parameters

- **value** The ULBulkCopyColumnMapping object to be removed from the collection.

See also

- "ULBulkCopyColumnMapping class" on page 49

RemoveAt method

Removes the mapping at the specified index from the collection.

Visual Basic syntax

```
Public Shadows Sub RemoveAt(ByVal index As Integer)
```

C# syntax

```
public new void RemoveAt(int index)
```

Parameters

- **index** The zero-based index of the ULBulkCopyColumnMapping object to be removed from the collection.

this property

Gets the ULBulkCopyColumnMapping object at the specified index.

Visual Basic syntax

```
Public ReadOnly Property Item(
    ByVal index As Integer
) As ULBulkCopyColumnMapping
```

C# syntax

```
public ULBulkCopyColumnMapping this[int index] {get;}
```

Parameters

- **index** The zero-based index of the ULBulkCopyColumnMapping object to find.

Remarks

An ULBulkCopyColumnMapping object is returned.

See also

- "ULBulkCopyColumnMapping class" on page 49

ULCommand class

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

Visual Basic syntax

```
Public NotInheritable Class ULCommand
    Inherits System.Data.Common.DbCommand
    Implements System.ICloneable
```

C# syntax

```
public sealed class ULCommand :
    System.Data.Common.DbCommand,
    System.ICloneable
```

Base classes

- System.Data.Common.DbCommand
- System.ICloneable

Members

All members of ULCommand class, including all inherited members.

Name	Description
"ULCommand constructor"	Initializes a ULCommand object.

Name	Description
"BeginExecuteNonQuery method"	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand.
"BeginExecuteReader method"	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, and retrieves the result set.
"Cancel method"	This method is not supported in UltraLite.NET.
"CreateParameter method"	Provides a ULParameter object for supplying parameters to ULCommand objects.
"EndExecuteNonQuery method"	Finishes asynchronous execution of a SQL statement.
"EndExecuteReader method"	Finishes asynchronous execution of a SQL statement, returning the requested ULDataReader.
ExecuteDbDataReader method (Inherited from System.Data.Common.DbCommand)	Executes the command text against the connection.
"ExecuteNonQuery method"	Executes a statement that does not return a result set, such as a SQL INSERT, DELETE, or UPDATE statement.
"ExecuteReader method"	Executes a SQL SELECT statement and returns the result set.

Name	Description
“ExecuteResultSet method”	**UL Ext:** Executes a SQL SELECT statement and returns the result set as a ULResultSet.
“ExecuteScalar method”	Executes a SQL SELECT statement and returns a single value.
“ExecuteTable method”	**UL Ext:** Retrieves in a ULTable a database table for direct manipulation.
“Prepare method”	Pre-compiles and stores the SQL statement of this command.
“CommandText property”	Specifies the text of the SQL statement or the name of the table when the ULCommand.CommandType is System.Data.CommandType.TableDirect.
“CommandTimeout property”	This feature is not supported by UltraLite.NET.
“CommandType property”	Specifies the type of command to be executed.
“Connection property”	The connection object on which to execute the ULCommand object.
“DesignTimeVisible property”	Indicates if the ULCommand should be visible in a customized Windows Form Designer control.

Name	Description
"IndexName property"	**UL Ext:** Specifies the name of the index to open (sort) the table with when the ULCommand.CommandType is System.Data.CommandType.TableDirect.
"Parameters property"	Specifies the parameters for the current statement.
"Plan property"	**UL Ext:** Returns the access plan UltraLite.NET uses to execute a query.
"Transaction property"	Specifies the ULTransaction in which the ULCommand executes.
"UpdatedRowSource property"	Specifies how command results are applied to the DataRow when used by the Update method of the ULDataAdapter.

Remarks

This object can be used to execute a statement or query efficiently multiple times.

ULCommand objects can be created directly, or with the ULConnection.CreateCommand method. This method ensures that the command has the correct transaction for executing statements on the given connection.

The ULCommand.Transaction method must be reset after the current transaction is committed or rolled back.

ULCommand features the following methods for executing commands on an UltraLite.NET database:

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

Method	Description
ULCommand.ExecuteReader()	Executes a SQL SELECT statement and returns the result set in a ULDataReader. Use this method for creating read-only result sets.
ULCommand.ExecuteResultSet()	UL Ext: Executes a SQL SELECT statement and returns the result set in a ULResultSet. Use this method for creating mutable result sets.
ULCommand.ExecuteScalar	Executes a SQL SELECT statement and returns a single value.
ULCommand.ExecuteTable()	UL Ext: Retrieves a database table in a ULTable for direct manipulation. The ULCommand.CommandText is interpreted as the name of the table and the ULCommand.IndexName can be used to specify a table sorting order. The ULCommand.CommandType must be System.Data.CommandType.TableDirect.

You can reset most properties, including the ULCommand.CommandText, and reuse the ULCommand object.

For resource management reasons, it is recommended that you explicitly dispose of commands when you are done with them. In C#, you may use a using statement to automatically call the System.ComponentModel.Component.Dispose() method or explicitly call the System.ComponentModel.Component.Dispose() method. In Visual Basic, you always explicitly call the System.ComponentModel.Component.Dispose() method.

See also

- “CreateCommand method” on page 123
- “Transaction property” on page 98
- “ExecuteNonQuery method” on page 84
- “ExecuteReader method” on page 85
- “ExecuteResultSet method” on page 87
- “ExecuteScalar method” on page 90
- “ExecuteTable method” on page 91
- “ULTable class” on page 394
- “CommandText property” on page 94
- “CommandType property” on page 95
- “IndexName property” on page 97
- “ULResultSet class” on page 334
- “ULDataReader class” on page 220
- System.Data.Common.DbCommand

ULCommand constructor

Initializes a ULCommand object.

Overload list

Name	Description
"ULCommand() constructor"	Initializes a ULCommand object.
"ULCommand(string) constructor"	Initializes a ULCommand object with the specified command text.
"ULCommand(string, ULConnection) constructor"	Initializes a ULCommand object with the specified command text and connection.
"ULCommand(string, ULConnection, ULTransaction) constructor"	Initializes a ULCommand object with the specified command text, connection, and transaction.

ULCommand() constructor

Initializes a ULCommand object.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ULCommand()
```

Remarks

The ULCommand object needs to have the ULCommand.CommandText, ULCommand.Connection, and ULCommand.Transaction properties set before a statement can be executed.

See also

- "CreateCommand method" on page 123
- "CommandText property" on page 94
- "Connection property" on page 96
- "Transaction property" on page 98

ULCommand(string) constructor

Initializes a ULCommand object with the specified command text.

Visual Basic syntax

```
Public Sub New(ByVal cmdText As String)
```

C# syntax

```
public ULCommand(string cmdText)
```

Parameters

- **cmdText** The text of the SQL statement or name of the table when the ULCommand.CommandType is System.Data.CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.

Remarks

The ULCommand object needs to have the ULCommand.Connection and ULCommand.Transaction set before a statement can be executed.

See also

- "CreateCommand method" on page 123
- "ULCommand constructor" on page 68
- "Connection property" on page 96
- "Transaction property" on page 98
- "CommandType property" on page 95
- System.Data.CommandType

ULCommand(string, ULConnection) constructor

Initializes a ULCommand object with the specified command text and connection.

Visual Basic syntax

```
Public Sub New(
    ByVal cmdText As String,
    ByVal connection As ULConnection
)
```

C# syntax

```
public ULCommand(string cmdText, ULConnection connection)
```

Parameters

- **cmdText** The text of the SQL statement or name of the table when the ULCommand.CommandType is System.Data.CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.
- **connection** The ULConnection object representing the current connection.

Remarks

The ULCommand object may need to have the ULCommand.Transaction set before a statement can be executed.

See also

- "ULConnection class" on page 110
- "CreateCommand method" on page 123
- "ULCommand constructor" on page 68
- "Transaction property" on page 98
- "CommandType property" on page 95
- System.Data.CommandType

ULCommand(string, ULConnection, ULTransaction) constructor

Initializes a ULCommand object with the specified command text, connection, and transaction.

Visual Basic syntax

```
Public Sub New(
    ByVal cmdText As String,
    ByVal connection As ULConnection,
    ByVal transaction As ULTransaction
)
```

C# syntax

```
public ULCommand(
    string cmdText,
    ULConnection connection,
    ULTransaction transaction
)
```

Parameters

- **cmdText** The text of the SQL statement or name of the table when the ULCommand.CommandType is System.Data.CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.
- **connection** The ULConnection object representing the current connection.
- **transaction** The ULTransaction in which the ULCommand executes.

See also

- "CreateCommand method" on page 123
- "ULCommand constructor" on page 68
- "CommandType property" on page 95
- "ULTransaction class" on page 430
- System.Data.CommandType

BeginExecuteNonQuery method

Initiates the asynchronous execution of a SQL statement that is described by this ULCommand.

Overload list

Name	Description
"BeginExecuteNonQuery() method"	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand.
"BeginExecuteNonQuery(AsyncCallback, object) method"	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, given a callback procedure and state information.

BeginExecuteNonQuery() method

Initiates the asynchronous execution of a SQL statement that is described by this ULCommand.

Visual Basic syntax

```
Public Function BeginExecuteNonQuery() As IAsyncResult
```

C# syntax

```
public IAsyncResult BeginExecuteNonQuery()
```

Returns

A System.IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking EndExecuteNonQuery(IAsyncResult), which returns the number of affected rows.

Exceptions

- **"ULException class"** Any error that occurred while executing the command text.

See also

- "EndExecuteNonQuery method" on page 77
- System.IAsyncResult

BeginExecuteNonQuery(AsyncCallback, object) method

Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, given a callback procedure and state information.

Visual Basic syntax

```
Public Function BeginExecuteNonQuery(
    ByVal callback As AsyncCallback,
    ByVal stateObject As Object
) As IAsyncResult
```

C# syntax

```
public IAsyncResult BeginExecuteNonQuery(
    AsyncCallback callback,
```



```
    object stateObject
)
```

Parameters

- **callback** An System.AsyncCallback delegate that is invoked when the command's execution has completed. Pass null (Nothing in Microsoft Visual Basic) to indicate that no callback is required.
- **stateObject** A user-defined state object that is passed to the callback procedure. Retrieve this object from within the callback procedure using the System.IAsyncResult.AsyncState property.

Returns

A System.IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking EndExecuteNonQuery(IAsyncResult), which returns the number of affected rows.

Exceptions

- **"ULException class"** Any error that occurred while executing the command text.

See also

- "EndExecuteNonQuery method" on page 77
- System.IAsyncResult.AsyncState

BeginExecuteReader method

Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, and retrieves the result set.

Overload list

Name	Description
"BeginExecuteReader() method"	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, and retrieves the result set.
"BeginExecuteReader(AsyncCallback, object) method"	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, and retrieves the result set, given a callback procedure and state information.
"BeginExecuteReader(AsyncCallback, object, CommandBehavior) method"	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, using one of the CommandBehavior values, and retrieves the result set, given a callback procedure and state information.
"BeginExecuteReader(CommandBehavior) method"	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, using one of the CommandBehavior values, and retrieves the result set.

BeginExecuteReader() method

Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, and retrieves the result set.

Visual Basic syntax

```
Public Function BeginExecuteReader() As IAsyncResult
```

C# syntax

```
public IAsyncResult BeginExecuteReader()
```

Returns

An System.IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns an ULDataReader instance that can be used to retrieve the returned rows.

Exceptions

- **"ULException class"** Any error that occurred while executing the command text.

See also

- "EndExecuteReader method" on page 81
- "ULDataReader class" on page 220
- System.IAsyncResult

BeginExecuteReader(AsyncCallback, object) method

Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, and retrieves the result set, given a callback procedure and state information.

Visual Basic syntax

```
Public Function BeginExecuteReader(
    ByVal callback As AsyncCallback,
    ByVal stateObject As Object
) As IAsyncResult
```

C# syntax

```
public IAsyncResult BeginExecuteReader(
    AsyncCallback callback,
    object stateObject
)
```

Parameters

- **callback** An System.AsyncCallback delegate that is invoked when the command's execution has completed. Pass null (Nothing in Microsoft Visual Basic) to indicate that no callback is required.

- **stateObject** A user-defined state object that is passed to the callback procedure. Retrieve this object from within the callback procedure using the System.IAsyncResult.AsyncState property.

Returns

An System.IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns an ULDataReader instance that can be used to retrieve the returned rows.

Exceptions

- **"ULException class"** Any error that occurred while executing the command text.

See also

- "ULDataReader class" on page 220
- "EndExecuteReader method" on page 81
- System.AsyncCallback

BeginExecuteReader(AsyncCallback, object, CommandBehavior) method

Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, using one of the CommandBehavior values, and retrieves the result set, given a callback procedure and state information.

Visual Basic syntax

```
Public Function BeginExecuteReader(
    ByVal callback As AsyncCallback,
    ByVal stateObject As Object,
    ByVal cmdBehavior As CommandBehavior
) As IAsyncResult
```

C# syntax

```
public IAsyncResult BeginExecuteReader(
    AsyncCallback callback,
    object stateObject,
    CommandBehavior cmdBehavior
)
```

Parameters

- **callback** A System.AsyncCallback delegate that is invoked when the command's execution has completed. Pass null (Nothing in Microsoft Visual Basic) to indicate that no callback is required.
- **stateObject** A user-defined state object that is passed to the callback procedure. Retrieve this object from within the callback procedure using the System.IAsyncResult.AsyncState property.
- **cmdBehavior** A bitwise combination of System.Data.CommandBehavior flags describing the results of the query and its effect on the connection. UltraLite.NET respects only the System.Data.CommandBehavior.Default, System.Data.CommandBehavior.CloseConnection, and System.Data.CommandBehavior.SchemaOnly flags.

Returns

A System.IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns an ULDataReader instance that can be used to retrieve the returned rows.

Exceptions

- **"ULException class"** Any error that occurred while executing the command text.

See also

- "EndExecuteReader method" on page 81
- "ULDataReader class" on page 220
- System.AsyncCallback

BeginExecuteReader(CommandBehavior) method

Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, using one of the CommandBehavior values, and retrieves the result set.

Visual Basic syntax

```
Public Function BeginExecuteReader(
    ByVal cmdBehavior As CommandBehavior
) As IAsyncResult
```

C# syntax

```
public IAsyncResult BeginExecuteReader(CommandBehavior cmdBehavior)
```

Parameters

- **cmdBehavior** A bitwise combination of System.Data.CommandBehavior flags describing the results of the query and its effect on the connection. UltraLite.NET respects only the System.Data.CommandBehavior.Default, System.Data.CommandBehavior.CloseConnection, and System.Data.CommandBehavior.SchemaOnly flags.

Returns

An System.IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns an ULDataReader instance that can be used to retrieve the returned rows.

Exceptions

- **"ULException class"** Any error that occurred while executing the command text.

See also

- "EndExecuteReader method" on page 81
- "ULDataReader class" on page 220
- System.Data.CommandBehavior

Cancel method

This method is not supported in UltraLite.NET.

Visual Basic syntax

```
Public Overrides Sub Cancel()
```

C# syntax

```
public override void Cancel()
```

Remarks

This method does nothing. UltraLite.NET commands cannot be interrupted while they are executing.

CreateParameter method

Provides a ULParameter object for supplying parameters to ULCommand objects.

Visual Basic syntax

```
Public Shadows Function CreateParameter() As ULParameter
```

C# syntax

```
public new ULParameter CreateParameter()
```

Returns

A new parameter, as a ULParameter object.

Remarks

Some SQL statements can take parameters, indicated in the text of a statement by a question mark (?). The CreateParameter method provides a ULParameter object. You can set properties on the ULParameter to specify the value for the parameter.

This is the strongly-typed version of System.Data.IDbCommand.CreateParameter and System.Data.Common.DbCommand.CreateParameter.

See also

- "ULParameter class" on page 301
- System.Data.IDbCommand.CreateParameter

EndExecuteNonQuery method

Finishes asynchronous execution of a SQL statement.

Visual Basic syntax

```
Public Function EndExecuteNonQuery(
    ByVal asyncResult As IAsyncResult
) As Integer
```

C# syntax

```
public int EndExecuteNonQuery(IAsyncResult asyncResult)
```

Parameters

- **asyncResult** The System.IAsyncResult returned by the call to BeginExecuteNonQuery.

Returns

The number of rows affected (the same behavior as ExecuteNonQuery).

Exceptions

- **ArgumentException** The asyncResult parameter is null (Nothing in Microsoft Visual Basic).
- **InvalidOperationException** The EndExecuteNonQuery(IAsyncResult) was called more than once for a single command execution, or the method was mismatched against its execution method.

Remarks

You must call EndExecuteNonQuery once for every call to BeginExecuteNonQuery. The call must be after BeginExecuteNonQuery has returned. ADO.NET is not thread safe; it is your responsibility to ensure that BeginExecuteNonQuery has returned. The System.IAsyncResult passed to EndExecuteNonQuery must be the same as the one returned from the BeginExecuteNonQuery call that is being completed. It is an error to call EndExecuteNonQuery to end a call to BeginExecuteReader, and vice versa.

If an error occurs while executing the command, the exception is thrown when EndExecuteNonQuery is called.

There are four ways to wait for execution to complete:

Call EndExecuteNonQuery Calling EndExecuteNonQuery blocks until the command completes. For example:

```
' Visual Basic
Dim cmd As ULCommand = new ULCommand( _
    "UPDATE Departments" _
    + " SET DepartmentName = 'Engineering'" _
    + " WHERE DepartmentID=100", _
    conn _
)

Dim res As IAsyncResult res = _
    cmd.BeginExecuteNonQuery()

' Perform other work.

' This blocks until the command completes.
Dim rowCount As Integer = _
    cmd.EndExecuteNonQuery( res )
```


The following code is the C# language equivalent:

```
// C#
ULCommand cmd = new ULCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn
);

IAsyncResult res = cmd.BeginExecuteNonQuery();

// Perform other work.

// This blocks until the command completes.
int rowCount = cmd.EndExecuteNonQuery( res );
```

Poll the IsCompleted property of the IAsyncResult You can poll the IsCompleted property of the IAsyncResult. For example:

```
' Visual Basic
Dim cmd As ULCommand = new ULCommand( _
    "UPDATE Departments" _
    + " SET DepartmentName = 'Engineering'" _
    + " WHERE DepartmentID=100", _
    conn _
)

Dim res As IAsyncResult res = _
    cmd.BeginExecuteNonQuery()
While( !res.IsCompleted )
    ' Perform other work.
End While

' This blocks until the command completes.
Dim rowCount As Integer = _
    cmd.EndExecuteNonQuery( res )
```

The following code is the C# language equivalent:

```
// C#
ULCommand cmd = new ULCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn
);

IAsyncResult res = cmd.BeginExecuteNonQuery();
while( !res.IsCompleted ) {
    // Perform other work.
}

// This blocks until the command completes.
int rowCount = cmd.EndExecuteNonQuery( res );
```

Use the IAsyncResult.AsyncWaitHandle property to get a synchronization object You can use the IAsyncResult.AsyncWaitHandle property to get a synchronization object, and wait on that. For example:

```
' Visual Basic
Dim cmd As ULCommand = new ULCommand( _
    "UPDATE Departments" _
```

```
    + " SET DepartmentName = 'Engineering'" _
    + " WHERE DepartmentID=100", _
    conn _
)
Dim res As IAsyncResult res = _
    cmd.BeginExecuteNonQuery()

' Perform other work.

Dim wh As WaitHandle = res.AsyncWaitHandle
wh.WaitOne()
' This does not block because the command is finished.
Dim rowCount As Integer = _
  cmd.EndExecuteNonQuery( res )
```

The following code is the C# language equivalent:

```
// C#
ULCommand cmd = new ULCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn
  );
IAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
WaitHandle wh = res.AsyncWaitHandle;
wh.WaitOne();
// This does not block because the command is finished.
int rowCount = cmd.EndExecuteNonQuery( res );
```

Specify a callback function when calling BeginExecuteNonQuery You can specify a callback function when calling BeginExecuteNonQuery. For example:

```
' Visual Basic
Private Sub callbackFunction(ByVal ar As IAsyncResult)
    Dim cmd As ULCommand = _
        CType(ar.AsyncState, ULCommand)
    ' This won't block since the command has completed.
    Dim rowCount As Integer = _
        cmd.EndExecuteNonQuery( res )
End Sub

' Elsewhere in the code
Private Sub DoStuff()
    Dim cmd As ULCommand = new ULCommand( _
        "UPDATE Departments" _
        + " SET DepartmentName = 'Engineering'" _
        + " WHERE DepartmentID=100", _
        conn _
    )
    Dim res As IAsyncResult = _
        cmd.BeginExecuteNonQuery( _
        callbackFunction, cmd _
    )
    ' Perform other work.  The callback function
    ' is called when the command completes.
End Sub
```

The following code is the C# language equivalent:

```
// C#
private void callbackFunction( IAsyncResult ar )
```



```
{
    ULCommand cmd = (ULCommand) ar.AsyncState;
    // This won't block since the command has completed.
    int rowCount = cmd.EndExecuteNonQuery();
}

// Elsewhere in the code
private void DoStuff()
{
    ULCommand cmd = new ULCommand(
        "UPDATE Departments"
        + " SET DepartmentName = 'Engineering'"
        + " WHERE DepartmentID=100",
        conn
    );
    IAsyncResult res = cmd.BeginExecuteNonQuery(
        callbackFunction, cmd
    );
    // Perform other work.  The callback function
    // is called when the command completes.
}
```

The callback function executes in a separate thread, so the usual caveats related to updating the user interface in a threaded program apply.

See also

- "BeginExecuteNonQuery method" on page 71
- System.IAsyncResult

EndExecuteReader method

Finishes asynchronous execution of a SQL statement, returning the requested ULDataReader.

Visual Basic syntax

```
Public Function EndExecuteReader(
    ByVal asyncResult As IAsyncResult
) As ULDataReader
```

C# syntax

```
public ULDataReader EndExecuteReader(IAsyncResult asyncResult)
```

Parameters

- **asyncResult** The System.IAsyncResult returned by the call to BeginExecuteReader.

Returns

An ULDataReader object that can be used to retrieve the requested rows (the same behavior as ExecuteReader).

Exceptions

- **ArgumentException** The asyncResult parameter is null (Nothing in Microsoft Visual Basic)

- **InvalidOperationException** The EndExecuteReader was called more than once for a single command execution, or the method was mismatched against its execution method.

Remarks

You must call EndExecuteReader once for every call to BeginExecuteReader. The call must be after BeginExecuteReader has returned. ADO.NET is not thread safe; it is your responsibility to ensure that BeginExecuteReader has returned. The System.IAsyncResult passed to EndExecuteReader must be the same as the one returned from the BeginExecuteReader call that is being completed. It is an error to call EndExecuteReader to end a call to BeginExecuteNonQuery, and vice versa.

If an error occurs while executing the command, the exception is thrown when EndExecuteReader is called.

There are four ways to wait for execution to complete:

Call EndExecuteReader Calling EndExecuteReader blocks until the command completes. For example:

```
' Visual Basic
Dim cmd As ULCommand = new ULCommand( _
    "SELECT * FROM Departments", conn _
)
Dim res As IAsyncResult res = _
    cmd.BeginExecuteReader()
' Perform other work
' This blocks until the command completes.
Dim reader As ULDataReader = _
    cmd.EndExecuteReader( res )
```

The following code is the C# language equivalent:

```
// C#
ULCommand cmd = new ULCommand(
    "SELECT * FROM Departments", conn
);
IAsyncResult res = cmd.BeginExecuteReader();

// Perform other work
// This blocks until the command completes
ULDataReader reader = cmd.EndExecuteReader( res );
```

Poll the IsCompleted property of the IAsyncResult You can poll the IsCompleted property of the IAsyncResult. For example:

```
' Visual Basic
Dim cmd As ULCommand = new ULCommand( _
    "SELECT * FROM Departments", conn _
)
Dim res As IAsyncResult res = _
    cmd.BeginExecuteReader()
While( !res.IsCompleted )
    ' Perform other work
End While
' This blocks until the command completes.
Dim reader As ULDataReader = _
    cmd.EndExecuteReader( res )


// C#
```



```
ULCommand cmd = new ULCommand(
    "SELECT * FROM Departments", conn
);
IAsyncResult res = cmd.BeginExecuteReader();
while( !res.IsCompleted ) {
    // Perform other work.
}
// This blocks until the command completes.
ULDataReader reader = cmd.EndExecuteReader( res );
```

Use the IAsyncResult.AsyncWaitHandle property to get a synchronization object You can use the IAsyncResult.AsyncWaitHandle property to get a synchronization object, and wait on that. For example:

```
' Visual Basic
Dim cmd As ULCommand = new ULCommand( _
    "SELECT * FROM Departments", conn _
)
Dim res As IAsyncResult res = _
    cmd.BeginExecuteReader()
' Perform other work.
Dim wh As WaitHandle = res.AsyncWaitHandle
wh.WaitOne()
' This does not block because the command is finished.
Dim reader As ULDataReader = _
    cmd.EndExecuteReader( res )
```

The following code is the C# language equivalent:

```
// C#
ULCommand cmd = new ULCommand(
    "SELECT * FROM Departments", conn
);
IAsyncResult res = cmd.BeginExecuteReader();
// Perform other work.
WaitHandle wh = res.AsyncWaitHandle;
wh.WaitOne();
// This does not block because the command is finished.
ULDataReader reader = cmd.EndExecuteReader( res );
```

Specify a callback function when calling BeginExecuteReader You can specify a callback function when calling BeginExecuteReader. For example:

```
' Visual Basic
Private Sub callbackFunction(ByVal ar As IAsyncResult)
    Dim cmd As ULCommand = _
        CType(ar.AsyncState, ULCommand)
    ' This won't block since the command has completed.
    Dim reader As ULDataReader = cmd.EndExecuteReader()
End Sub

' Elsewhere in the code
Private Sub DoStuff()
    Dim cmd As ULCommand = new ULCommand( _
        "SELECT * FROM Departments", conn _
    )
    Dim res As IAsyncResult = _
        cmd.BeginExecuteReader( _
        callbackFunction, cmd _
    )
    ' Perform other work.  The callback function
    ' is called when the command completes.
End Sub
```

The following code is the C# language equivalent:

```
// C#
private void callbackFunction( IAsyncResult ar )
{
    ULCommand cmd = (ULCommand) ar.AsyncState;
    // This won't block since the command has completed.
    ULDataReader reader = cmd.EndExecuteReader();
}

// Elsewhere in the code.
private void DoStuff()
{
    ULCommand cmd = new ULCommand(
        "SELECT * FROM Departments", conn
    );
    IAsyncResult res = cmd.BeginExecuteReader(callbackFunction, cmd);

    // Perform other work.  The callback function
    // is called when the command completes.
}
```

The callback function executes in a separate thread, so the usual caveats related to updating the user interface in a threaded program apply.

See also

- "BeginExecuteReader method" on page 73
- "ULDataReader class" on page 220
- System.IAsyncResult

ExecuteNonQuery method

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

Visual Basic syntax

```
Public Overrides Function ExecuteNonQuery() As Integer
```

C# syntax

```
public override int ExecuteNonQuery()
```

Returns

The number of rows affected.

Exceptions

- **"ULException class"** A SQL error occurred.
- **InvalidOperationException** The command is in an invalid state. Either the ULCommand.Connection is missing or closed, the ULCommand.Transaction value does not match the current transaction state of the connection, or the ULCommand.CommandText is invalid.

Remarks

The statement is the current ULCommand object, with the ULCommand.CommandText and ULCommand.Parameters as needed.

For UPDATE, INSERT, and DELETE statements, the return value is the number of rows affected by the command. For all other types of statements, and for rollbacks, the return value is -1.

The ULCommand.CommandType cannot be System.Data.CommandType.TableDirect.

See also

- "CommandText property" on page 94
- "CommandType property" on page 95
- "Connection property" on page 96
- "Parameters property" on page 97
- "Transaction property" on page 98
- System.Data.CommandType

ExecuteReader method

Executes a SQL SELECT statement and returns the result set.

Overload list

Name	Description
"ExecuteReader() method"	Executes a SQL SELECT statement and returns the result set.
"ExecuteReader(CommandBehavior) method"	Executes a SQL SELECT statement with the specified command behavior and returns the result set.

ExecuteReader() method

Executes a SQL SELECT statement and returns the result set.

Visual Basic syntax

```
Public Shadows Function ExecuteReader() As ULDataReader
```

C# syntax

```
public new ULDataReader ExecuteReader()
```

Returns

The result set as a ULDataReader object.

Exceptions

- **"ULException class"** A SQL error occurred.

- **InvalidOperationException** The command is in an invalid state. Either the ULCommand.Connection is missing or closed, the ULCommand.Transaction value does not match the current transaction state of the connection, or the ULCommand.CommandText is invalid.

Remarks

The statement is the current ULCommand object, with the ULCommand.CommandText and any ULCommand.Parameters as required. The ULDataReader object is a read-only result set. For editable result sets, use ULCommand.ExecuteResultSet(), ULCommand.ExecuteTable(), or a ULDataAdapter.

If the ULCommand.CommandType is System.Data.CommandType.TableDirect, ExecuteReader performs an ULCommand.ExecuteTable() and returns a ULTable downcast as a ULDataReader.

SELECT statements are marked as read-only by default for performance reasons. If the query is going to be used to make updates, the statement must end with " FOR UPDATE".

This is the strongly-typed version of System.Data.IDbCommand.ExecuteReader() and System.Data.Common.DbCommand.ExecuteReader().

See also

- "CommandText property" on page 94
- "CommandType property" on page 95
- "Connection property" on page 96
- "Parameters property" on page 97
- "ExecuteResultSet method" on page 87
- "ExecuteTable method" on page 91
- "Transaction property" on page 98
- "ULDataAdapter class" on page 196
- "ULDataReader class" on page 220
- "ULTable class" on page 394
- System.Data.CommandType

ExecuteReader(CommandBehavior) method

Executes a SQL SELECT statement with the specified command behavior and returns the result set.

Visual Basic syntax

```
Public Shadows Function ExecuteReader(
    ByVal cmdBehavior As CommandBehavior
) As ULDataReader
```

C# syntax

```
public new ULDataReader ExecuteReader(CommandBehavior cmdBehavior)
```

Parameters

- **cmdBehavior** A bitwise combination of System.Data.CommandBehavior flags describing the results of the query and its effect on the connection. UltraLite.NET respects only the

System.Data.CommandBehavior.Default, System.Data.CommandBehavior.CloseConnection, and System.Data.CommandBehavior.SchemaOnly flags.

Returns

The result set as a ULDataReader object.

Exceptions

- **"ULException class"** A SQL error occurred.
- **InvalidOperationException** The command is in an invalid state. Either the ULCommand.Connection is missing or closed, the ULCommand.Transaction value does not match the current transaction state of the connection, or the ULCommand.CommandText is invalid.

Remarks

The statement is the current ULCommand object, with the ULCommand.CommandText and any ULCommand.Parameters as required. The ULDataReader object is a read-only result set. For editable result sets, use ULCommand.ExecuteResultSet(CommandBehavior), ULCommand.ExecuteTable(CommandBehavior), or a ULDataAdapter.

If the ULCommand.CommandType is System.Data.CommandType.TableDirect, ExecuteReader performs an ULCommand.ExecuteTable(CommandBehavior) and returns a ULTable downcast as a ULDataReader.

SELECT statements are marked as read-only by default for performance reasons. If the query is going to be used to make updates, the statement must end with " FOR UPDATE".

This is the strongly-typed version of System.Data.IDbCommand.ExecuteReader(System.Data.CommandBehavior) and System.Data.Common.DbCommand.ExecuteReader(System.Data.CommandBehavior).

See also

- "CommandText property" on page 94
- "Connection property" on page 96
- "ExecuteReader method" on page 85
- "ExecuteResultSet method" on page 87
- "ExecuteTable method" on page 91
- "Parameters property" on page 97
- "Transaction property" on page 98
- "ULDataAdapter class" on page 196
- "ULDataReader class" on page 220
- "ULTable class" on page 394
- "ULDataReader class" on page 220
- System.Data.CommandType

ExecuteResultSet method

UL Ext: Executes a SQL SELECT statement and returns the result set as a ULResultSet.

Overload list

Name	Description
"ExecuteResultSet() method"	**UL Ext:** Executes a SQL SELECT statement and returns the result set as a ULResultSet.
"ExecuteResultSet(CommandBehavior) method"	**UL Ext:** Executes a SQL SELECT statement with the specified command behavior and returns the result set as a ULResultSet.

ExecuteResultSet() method

UL Ext: Executes a SQL SELECT statement and returns the result set as a ULResultSet.

Visual Basic syntax

```
Public Function ExecuteResultSet() As ULResultSet
```

C# syntax

```
public ULResultSet ExecuteResultSet()
```

Returns

The result set as a ULResultSet object.

Exceptions

- **"ULException class"** A SQL error occurred.
- **InvalidOperationException** The command is in an invalid state. Either the ULCommand.Connection is missing or closed, the ULCommand.Transaction value does not match the current transaction state of the connection, or the ULCommand.CommandText is invalid.

Remarks

The statement is the current ULCommand object, with the ULCommand.CommandText and any ULCommand.Parameters as required. The ULResultSet object is an editable result set on which you can perform positioned updates and deletes. For fully editable result sets, use ULCommand.ExecuteTable() or a ULDataAdapter.

If the ULCommand.CommandType is System.Data.CommandType.TableDirect, ExecuteReader performs an ULCommand.ExecuteTable() and returns a ULTable downcast as a ULResultSet.

ExecuteResultSet supports positioned updates and deletes with Dynamic SQL.

See also

- "ULCommand class" on page 64
- "CommandText property" on page 94
- "Parameters property" on page 97
- "CommandType property" on page 95
- "ULResultSet class" on page 334
- "ULTable class" on page 394
- System.Data.CommandType

Example

```
cmd.CommandText = "SELECT id, season, price FROM OurProducts";
ULResultSet rs = cmd.ExecuteResultSet();
while( rs.Read() ) {
    string season = rs.GetString( 1 );
    double price = rs.GetDouble( 2 );
    if( season.Equals( "summer" ) ) {
        rs.UpdateBegin();
        rs.SetDouble( 2, price * .5 );
        rs.Update();
    }
    if( season.Equals( "discontinued" ) ) {
        rs.Delete();
    }
}
rs.Close();
```

ExecuteResultSet(CommandBehavior) method

UL Ext: Executes a SQL SELECT statement with the specified command behavior and returns the result set as a ULResultSet.

Visual Basic syntax

```
Public Function ExecuteResultSet(
    ByVal cmdBehavior As CommandBehavior
) As ULResultSet
```

C# syntax

```
public ULResultSet ExecuteResultSet(CommandBehavior cmdBehavior)
```

Parameters

- **cmdBehavior** A bitwise combination of System.Data.CommandBehavior flags describing the results of the query and its effect on the connection. UltraLite.NET respects only the System.Data.CommandBehavior.Default, System.Data.CommandBehavior.CloseConnection, and System.Data.CommandBehavior.SchemaOnly flags.

Returns

The result set as a ULResultSet object.

Exceptions

- **"ULException class"** A SQL error occurred.

- **InvalidOperationException** The command is in an invalid state. Either the ULCommand.Connection is missing or closed, the ULCommand.Transaction value does not match the current transaction state of the connection, or the ULCommand.CommandText is invalid.

Remarks

The statement is the current ULCommand object, with the ULCommand.CommandText and any ULCommand.Parameters as required. The ULResultSet object is an editable result set on which you can perform positioned updates and deletes. For fully editable result sets, use ULCommand.ExecuteTable(CommandBehavior) or a ULDataAdapter.

If the ULCommand.CommandType is System.Data.CommandType.TableDirect, ExecuteReader performs an ULCommand.ExecuteTable(CommandBehavior) and returns a ULTable downcast as a ULResultSet.

ExecuteResultSet supports positioned updates and deletes with Dynamic SQL.

See also

- "ULResultSet class" on page 334
- "CommandText property" on page 94
- "CommandType property" on page 95
- "Connection property" on page 96
- "ExecuteReader method" on page 85
- "ExecuteTable method" on page 91
- "Parameters property" on page 97
- "Transaction property" on page 98
- "ULDataAdapter class" on page 196
- "ULTable class" on page 394
- System.Data.CommandType

ExecuteScalar method

Executes a SQL SELECT statement and returns a single value.

Visual Basic syntax

```
Public Overrides Function ExecuteScalar() As Object
```

C# syntax

```
public override object ExecuteScalar()
```

Returns

The first column of the first row in the result set, or a null reference (Nothing in Visual Basic) if the result set is empty.

Exceptions

- **"ULException class"** A SQL error occurred.

- **InvalidOperationException** The command is in an invalid state. Either the ULCommand.Connection is missing or closed, the ULCommand.Transaction value does not match the current transaction state of the connection, or the ULCommand.CommandText is invalid.

Remarks

The statement is the current ULCommand object, with the ULCommand.CommandText and any ULCommand.Parameters as required.

If this method is called on a query that returns multiple rows and columns, only the first column of the first row is returned.

If the ULCommand.CommandType is System.Data.CommandType.TableDirect, ExecuteScalar performs an ULCommand.ExecuteTable() and returns the first column of the first row.

SELECT statements are marked as read-only by default for performance reasons. If the query is going to be used to make updates, the statement must end with " FOR UPDATE".

See also

- "CommandText property" on page 94
- "CommandType property" on page 95
- "Connection property" on page 96
- "IndexName property" on page 97
- "Parameters property" on page 97
- "Transaction property" on page 98
- System.Data.CommandType

ExecuteTable method

UL Ext: Retrieves in a ULTable a database table for direct manipulation.

Overload list

Name	Description
"ExecuteTable() method"	**UL Ext:** Retrieves in a ULTable a database table for direct manipulation.
"ExecuteTable(CommandBehavior) method"	**UL Ext:** Retrieves, with the specified command behavior, a database table for direct manipulation.

ExecuteTable() method

UL Ext: Retrieves in a ULTable a database table for direct manipulation.

Visual Basic syntax

```
Public Function ExecuteTable() As ULTable
```

C# syntax

```
public ULTable ExecuteTable()
```

Returns

The table as a ULTable object.

Exceptions

- **"ULException class"** A SQL error occurred.
- **InvalidOperationException** The command is in an invalid state. Either the ULCommand.Connection is missing or closed, the ULCommand.Transaction value does not match the current transaction state of the connection, or the ULCommand.CommandText is invalid.

Remarks

The ULCommand.CommandText is interpreted as the name of the table and ULCommand.IndexName can be used to specify a table sorting order.

The ULCommand.CommandType must be set to System.Data.CommandType.TableDirect.

If the ULCommand.IndexName is a null reference (Nothing in Visual Basic), the primary key is used to open the table. Otherwise, the table is opened using the ULCommand.IndexName value as the name of the index by which to sort.

See also

- "CommandText property" on page 94
- "CommandType property" on page 95
- "Connection property" on page 96
- "IndexName property" on page 97
- "Transaction property" on page 98
- "ULTable class" on page 394
- System.Data.CommandBehavior

ExecuteTable(CommandBehavior) method

UL Ext: Retrieves, with the specified command behavior, a database table for direct manipulation.

Visual Basic syntax

```
Public Function ExecuteTable(
    ByVal cmdBehavior As CommandBehavior
) As ULTable
```

C# syntax

```
public ULTable ExecuteTable(CommandBehavior cmdBehavior)
```

Parameters

- **cmdBehavior** A bitwise combination of System.Data.CommandBehavior flags describing the results of the query and its effect on the connection. UltraLite.NET respects only the

System.Data.CommandBehavior.Default, System.Data.CommandBehavior.CloseConnection, and System.Data.CommandBehavior.SchemaOnly flags.

Returns

The table as a ULTable object.

Exceptions

- **"ULException class"** A SQL error occurred.
- **InvalidOperationException** The command is in an invalid state. Either the ULCommand.Connection is missing or closed, the ULCommand.Transaction value does not match the current transaction state of the connection, or the ULCommand.CommandText is invalid.

Remarks

The ULCommand.CommandText is interpreted as the name of the table and ULCommand.IndexName can be used to specify a table sorting order.

The ULCommand.CommandType must be set to System.Data.CommandType.TableDirect.

If the ULCommand.IndexName is a null reference (Nothing in Visual Basic), the primary key is used to open the table. Otherwise, the table is opened using the ULCommand.IndexName value as the name of the index by which to sort.

See also

- "CommandText property" on page 94
- "CommandType property" on page 95
- "Connection property" on page 96
- "ExecuteTable method" on page 91
- "IndexName property" on page 97
- "Transaction property" on page 98
- System.Data.CommandBehavior

Prepare method

Pre-compiles and stores the SQL statement of this command.

Visual Basic syntax

```
Public Overrides Sub Prepare()
```

C# syntax

```
public override void Prepare()
```

Exceptions

- **"ULException class"** A SQL error occurred.

- **InvalidOperationException** The command is in an invalid state. Either the ULCommand.Connection is missing or closed, the ULCommand.Transaction value does not match the current transaction state of the connection, or the ULCommand.CommandText is invalid.

Remarks

Pre-compiling statements allows for the efficient re-use of statements when just the parameter values are changed. Changing any other property on this command unprepares the statement.

UltraLite.NET does not require you to explicitly prepare statements as all unprepared commands are prepared on calls to the various Execute methods.

See also

- "Connection property" on page 96
- "Transaction property" on page 98
- "CommandText property" on page 94

CommandText property

Specifies the text of the SQL statement or the name of the table when the ULCommand.CommandType is System.Data.CommandType.TableDirect.

Visual Basic syntax

```
Public Overrides Property CommandText As String
```

C# syntax

```
public override string CommandText {get;set;}
```

Remarks

For parameterized statements, use a question mark (?) placeholder to pass parameters.

A string specifying the text of the SQL statement or the name of the table. The default is an empty string (invalid command).

SELECT statements are marked as read-only by default for performance reasons. If the query is going to be used to make updates, the statement must end with " FOR UPDATE".

See also

- "ExecuteNonQuery method" on page 84
- "ExecuteReader method" on page 85
- "ExecuteResultSet method" on page 87
- "ExecuteScalar method" on page 90
- "ExecuteTable method" on page 91
- "CommandType property" on page 95
- System.Data.CommandType

Example

The following example demonstrates the use of the parameterized placeholder:

```
' Visual Basic
myCmd.CommandText = "SELECT * FROM Customers WHERE CustomerID = ?"
```

The following code is the C# language equivalent:

```
// C#
myCmd.CommandText = "SELECT * FROM Customers WHERE CustomerID = ?";
```

CommandTimeout property

This feature is not supported by UltraLite.NET.

Visual Basic syntax

```
Public Overrides Property CommandTimeout As Integer
```

C# syntax

```
public override int CommandTimeout {get;set;}
```

Exceptions

- **"ULException class"** Setting the value is not supported in UltraLite.NET.

Remarks

The value is always zero.

CommandType property

Specifies the type of command to be executed.

Visual Basic syntax

```
Public Overrides Property CommandType As CommandType
```

C# syntax

```
public override CommandType CommandType {get;set;}
```

Exceptions

- **ArgumentException** CommandType.StoredProcedure is not supported in UltraLite.NET.

Remarks

One of the System.Data.CommandType values. The default is System.Data.CommandType.Text.

Supported command types are as follows:

- System.Data.CommandType.TableDirect - **UL Ext:** When you specify this CommandType, the ULCommand.CommandText must be the name of a database table. You can also specify the index used to open (sort) the table with ULCommand.IndexName. Use ULCommand.ExecuteTable() or ULCommand.ExecuteReader() to access the table.

- System.Data.CommandType.Text - When you specify this CommandType, the ULCommand.CommandText must be a SQL statement or query. Use ULCommand.ExecuteNonQuery to execute a non-query SQL statement and use either ULCommand.ExecuteReader() or ULCommand.ExecuteScalar to execute a query.

See also

- "CommandText property" on page 94
- "IndexName property" on page 97
- "ExecuteTable method" on page 91
- "ExecuteReader method" on page 85
- "CommandText property" on page 94
- "ExecuteNonQuery method" on page 84
- "ExecuteReader method" on page 85
- "ExecuteScalar method" on page 90
- System.Data.CommandType

Connection property

The connection object on which to execute the ULCommand object.

Visual Basic syntax

```
Public Shadows Property Connection As ULConnection
```

C# syntax

```
public new ULConnection Connection {get;set;}
```

Remarks

The ULConnection object on which to execute the command.

ULCommand objects must have an open connection before they can be executed.

The default is a null reference (Nothing in Visual Basic).

This is the strongly-typed version of System.Data.IDbCommand.Connection and System.Data.Common.DbCommand.Connection.

See also

- "ULConnection class" on page 110
- System.Data.IDbCommand.Connection

DesignTimeVisible property

Indicates if the ULCommand should be visible in a customized Windows Form Designer control.

Visual Basic syntax

```
Public Overrides Property DesignTimeVisible As Boolean
```

C# syntax

```
public override bool DesignTimeVisible {get;set;}
```

Remarks

True if this ULCommand instance should be visible, false if this instance should not be visible. The default is false.

IndexName property

UL Ext: Specifies the name of the index to open (sort) the table with when the ULCommand.CommandType is System.Data.CommandType.TableDirect.

Visual Basic syntax

```
Public Property IndexName As String
```

C# syntax

```
public string IndexName {get;set;}
```

Remarks

A string specifying the name of the index. The default is a null reference (Nothing in Visual Basic), meaning the table is opened with its primary key.

See also

- "ExecuteTable method" on page 91
- "ExecuteReader method" on page 85
- "CommandType property" on page 95
- System.Data.CommandType

Parameters property

Specifies the parameters for the current statement.

Visual Basic syntax

```
Public ReadOnly Shadows Property Parameters As ULParameterCollection
```

C# syntax

```
public new ULParameterCollection Parameters {get;}
```

Remarks

A ULParameterCollection holding the parameters of the SQL statement. The default value is the empty collection.

Use question marks in ULCommand.CommandText to indicate parameters. The parameters in the collection are specified in the same order as the question mark placeholders. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText as there are parameters in this collection.

This is the strongly-typed version of System.Data.IDbCommand.Parameters and System.Data.Common.DbCommand.Parameters.

See also

- "ULParameterCollection class" on page 315
- "ULParameter class" on page 301
- "CommandText property" on page 94
- System.Data.IDbCommand.Connection

Plan property

UL Ext: Returns the access plan UltraLite.NET uses to execute a query.

Visual Basic syntax

```
Public ReadOnly Property Plan As String
```

C# syntax

```
public string Plan {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

This property is intended primarily for use during development.

A string containing the text-based description of the query execution plan.

Transaction property

Specifies the ULTransaction in which the ULCommand executes.

Visual Basic syntax

```
Public Shadows Property Transaction As ULTransaction
```

C# syntax

```
public new ULTransaction Transaction {get;set;}
```

Remarks

The ULTransaction in which the ULCommand executes. This should be the current transaction of the connection specified by the ULCommand.Connection. The default is a null reference (Nothing in Visual Basic).

If a command is reused after a transaction has been committed or rolled back, this property needs to be reset.

This is the strongly-typed version of System.Data.IDbCommand.Transaction and System.Data.Common.DbCommand.Transaction.

See also

- "BeginTransaction method" on page 117
- "ULTransaction class" on page 430
- "Connection property" on page 96
- System.Data.IDbCommand.Transaction

UpdatedRowSource property

Specifies how command results are applied to the DataRow when used by the Update method of the ULDataAdapter.

Visual Basic syntax

```
Public Overrides Property UpdatedRowSource As UpdateRowSource
```

C# syntax

```
public override UpdateRowSource UpdatedRowSource {get;set;}
```

Remarks

One of the System.Data.UpdateRowSource values. The default value is System.Data.UpdateRowSource.Both.

See also

- System.Data.UpdateRowSource

ULCommandBuilder class

Automatically generates single-table commands used to reconcile changes made to a System.Data.DataSet with the associated database.

Visual Basic syntax

```
Public Class ULCommandBuilder
    Inherits System.Data.Common.DbCommandBuilder
```

C# syntax

```
public class ULCommandBuilder : System.Data.Common.DbCommandBuilder
```

Base classes

- System.Data.Common.DbCommandBuilder

Members

All members of ULCommandBuilder class, including all inherited members.

Name	Description
"ULCommandBuilder constructor"	Initializes a ULCommandBuilder object.
ApplyParameterInfo method (Inherited from System.Data.Common.DbCommandBuilder)	Allows the provider implementation of the System.Data.Common.DbCommandBuilder class to handle additional parameter properties.
Dispose method (Inherited from System.Data.Common.DbCommandBuilder)	Releases the unmanaged resources used by the System.Data.Common.DbCommandBuilder and optionally releases the managed resources.
"GetDeleteCommand method"	Gets the automatically generated ULCommand object required to perform deletions on the database.
"GetInsertCommand method"	Gets the automatically generated ULCommand object required to perform insertions on the database.
GetParameterName method (Inherited from System.Data.Common.DbCommandBuilder)	Returns the name of the specified parameter in the format of @p#.
GetParameterPlaceholder method (Inherited from System.Data.Common.DbCommandBuilder)	Returns the placeholder for the parameter in the associated SQL statement.
GetSchemaTable method (Inherited from System.Data.Common.DbCommandBuilder)	Returns the schema table for the System.Data.Common.DbCommandBuilder.
"GetUpdateCommand method"	Gets the automatically generated ULCommand object required to perform updates on the database.

Name	Description
InitializeCommand method (Inherited from System.Data.Common.DbCommandBuilder)	Resets the System.Data.Common.DbCommand.CommandTimeout, System.Data.Common.DbCommand.Transaction, System.Data.Common.DbCommand.CommandType, and System.Data.UpdateRowSource properties on the System.Data.Common.DbCommand.
QuoteIdentifier method (Inherited from System.Data.Common.DbCommandBuilder)	Given an unquoted identifier in the correct catalog case, returns the correct quoted form of that identifier, including properly escaping any embedded quotes in the identifier.
RefreshSchema method (Inherited from System.Data.Common.DbCommandBuilder)	Clears the commands associated with this System.Data.Common.DbCommandBuilder.
RowUpdatingHandler method (Inherited from System.Data.Common.DbCommandBuilder)	Adds an event handler for the System.Data.OleDb.OleDbDataAdapter.RowUpdating event.
SetRowUpdatingHandler method (Inherited from System.Data.Common.DbCommandBuilder)	Registers the System.Data.Common.DbCommandBuilder to handle the System.Data.OleDb.OleDbDataAdapter.RowUpdating event for a System.Data.Common.DbDataAdapter
UnquoteIdentifier method (Inherited from System.Data.Common.DbCommandBuilder)	Given a quoted identifier, returns the correct unquoted form of that identifier, including properly un-escaping any embedded quotes in the identifier.
CatalogLocation property (Inherited from System.Data.Common.DbCommandBuilder)	Sets or gets the System.Data.Common.CatalogLocation for an instance of the System.Data.Common.DbCommandBuilder class.
CatalogSeparator property (Inherited from System.Data.Common.DbCommandBuilder)	Sets or gets a string used as the catalog separator for an instance of the System.Data.Common.DbCommandBuilder class.
ConflictOption property (Inherited from System.Data.Common.DbCommandBuilder)	Specifies which System.Data.ConflictOption is to be used by the System.Data.Common.DbCommandBuilder.
“DataAdapter property”	Gets or sets a ULDataAdapter object for which SQL statements are automatically generated.

Name	Description
QuotePrefix property (Inherited from System.Data.Common.DbCommandBuilder)	Gets or sets the beginning character or characters to use when specifying database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.
QuoteSuffix property (Inherited from System.Data.Common.DbCommandBuilder)	Gets or sets the beginning character or characters to use when specifying database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.
SchemaSeparator property (Inherited from System.Data.Common.DbCommandBuilder)	Gets or sets the character to be used for the separator between the schema identifier and any other identifiers.
SetAllValues property (Inherited from System.Data.Common.DbCommandBuilder)	Specifies whether all column values in an update statement are included or only changed ones.

Remarks

The ULDataAdapter does not automatically generate the SQL statements required to reconcile changes made to a System.Data.DataSet with the associated data source. However, you can create a ULCommandBuilder object to automatically generate SQL statements for single-table updates if you set the SelectCommand property of the ULDataAdapter. Then, any additional SQL statements that you do not set are generated by the ULCommandBuilder.

See also

- "ULCommand class" on page 64
- "ULConnection class" on page 110
- "ULDataAdapter class" on page 196
- System.Data.DataSet

Example

The following example uses the ULCommand, along with ULDataAdapter and ULConnection, to select rows from a data source. The example is passed a connection string, a query string that is a SQL SELECT statement, and a string that is the name of the database table. The example then creates a ULCommandBuilder.

```
' Visual Basic
Public Shared Function SelectULRows(ByVal connectionString As String, _
    ByVal queryString As String, ByVal tableName As String)

    Dim connection As ULConnection = New ULConnection(connectionString)
    Dim adapter As ULDataAdapter = New ULDataAdapter()

    adapter.SelectCommand = New ULCommand(queryString, connection)
```



```
        Dim builder As ULCommandBuilder = New ULCommandBuilder(adapter)

        connection.Open()

        Dim dataSet As DataSet = New DataSet()
        adapter.Fill(dataSet, tableName)

        'Insert code to modify data in DataSet.

        'Without the ULCommandBuilder this line would fail
        adapter.Update(dataSet, tableName)

        Return dataSet
    End Function
```

The following code is the C# language equivalent:

```
// C#
public static DataSet SelectULRows(string connectionString,
    string queryString, string tableName)
{
    using (ULConnection connection = new ULConnection(connectionString))
    {
        ULDataAdapter adapter = new ULDataAdapter();
        adapter.SelectCommand = new ULCommand(queryString, connection);
        ULCommandBuilder builder = new ULCommandBuilder(adapter);

        connection.Open();

        DataSet dataSet = new DataSet();
        adapter.Fill(dataSet, tableName);

           // Insert code to modify data in DataSet.

        // Without the ULCommandBuilder this line would fail
        adapter.Update(dataSet, tableName);

        return dataSet;
    }
}
```

ULCommandBuilder constructor

Initializes a ULCommandBuilder object.

Overload list

Name	Description
"ULCommandBuilder() constructor"	Initializes a ULCommandBuilder object.
"ULCommandBuilder(ULDataAdapter) constructor"	Initializes a ULCommandBuilder object with the specified ULDataAdapter object.

ULCommandBuilder() constructor

Initializes a ULCommandBuilder object.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ULCommandBuilder()
```

See also

- "ULCommandBuilder constructor" on page 103

ULCommandBuilder(ULDataAdapter) constructor

Initializes a ULCommandBuilder object with the specified ULDataAdapter object.

Visual Basic syntax

```
Public Sub New(ByVal adapter As ULDataAdapter)
```

C# syntax

```
public ULCommandBuilder(ULDataAdapter adapter)
```

Parameters

- **adapter** A ULDataAdapter object.

See also

- "ULDataAdapter class" on page 196

GetDeleteCommand method

Gets the automatically generated ULCommand object required to perform deletions on the database.

Overload list

Name	Description
"GetDeleteCommand() method"	Gets the automatically generated ULCommand object required to perform deletions on the database.
"GetDeleteCommand(bool) method"	Gets the automatically generated ULCommand object required to perform deletions on the database.

GetDeleteCommand() method

Gets the automatically generated ULCommand object required to perform deletions on the database.

Visual Basic syntax

```
Public Shadows Function GetDeleteCommand() As ULCommand
```

C# syntax

```
public new ULCommand GetDeleteCommand()
```

Returns

The automatically generated ULCommand object required to perform deletions.

Exceptions

- **InvalidOperationException** The DbCommandBuilder.DataAdapter has not been initialized.The DataAdapter.SelectCommand property has not been initialized.The DataAdapter.SelectCommand.Connection property has not been initialized.Dynamic SQL generation is not supported against multiple base tables.Dynamic SQL generation is not supported against a SelectCommand that contains duplicate columns.Dynamic SQL generation for the DeleteCommand is not supported against a SelectCommand that does not return any key column information.

Remarks

After the SQL statement is first generated, the application must explicitly call the DbCommandBuilder.RefreshSchema if it changes the ULDataAdapter.SelectCommand in any way. Otherwise, the GetDeleteCommand method still uses information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either the DbDataAdapter.Update(System.Data.DataSet) or the GetDeleteCommand method.

See also

- "ULCommand class" on page 64
- "SelectCommand property" on page 203
- System.Data.Common.DbCommandBuilder.DataAdapter

GetDeleteCommand(bool) method

Gets the automatically generated ULCommand object required to perform deletions on the database.

Visual Basic syntax

```
Public Shadows Function GetDeleteCommand(
    ByVal useColumnsForParameterNames As Boolean
) As ULCommand
```

C# syntax

```
public new ULCommand GetDeleteCommand(bool useColumnsForParameterNames)
```

Parameters

- **useColumnsForParameterNames** If true, generate parameter names matching column names if possible. If false, generate @p1, @p2, and so on.

Returns

The automatically generated ULCommand object required to perform deletions.

Exceptions

- **InvalidOperationException** The DbCommandBuilder.DataAdapter has not been initialized.The DataAdapter.SelectCommand property has not been initialized.The DataAdapter.SelectCommand.Connection property has not been initialized.Dynamic SQL generation is not supported against multiple base tables.Dynamic SQL generation is not supported against a SelectCommand that contains duplicate columns.Dynamic SQL generation for the DeleteCommand is not supported against a SelectCommand that does not return any key column information.

Remarks

After the SQL statement is first generated, the application must explicitly call the DbCommandBuilder.RefreshSchema if it changes the ULDataAdapter.SelectCommand in any way. Otherwise, the GetDeleteCommand method still uses information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either the DbDataAdapter.Update(System.Data.DataSet) or the GetDeleteCommand method.

See also

- "ULCommand class" on page 64
- "SelectCommand property" on page 203
- System.Data.Common.DbCommandBuilder.DataAdapter

GetInsertCommand method

Gets the automatically generated ULCommand object required to perform insertions on the database.

Overload list

Name	Description
"GetInsertCommand() method"	Gets the automatically generated ULCommand object required to perform insertions on the database.
"GetInsertCommand(bool) method"	Gets the automatically generated ULCommand object required to perform insertions on the database.

GetInsertCommand() method

Gets the automatically generated ULCommand object required to perform insertions on the database.

Visual Basic syntax

```
Public Shadows Function GetInsertCommand() As ULCommand
```

C# syntax

```
public new ULCommand GetInsertCommand()
```

Returns

The automatically generated ULCommand object required to perform insertions.

Exceptions

- **InvalidOperationException** The DbCommandBuilder.DataAdapter has not been initialized.The DataAdapter.SelectCommand property has not been initialized.The DataAdapter.SelectCommand.Connection property has not been initialized.Dynamic SQL generation for the InsertCommand is not supported against a SelectCommand that does not return any modifiable columns.Dynamic SQL generation is not supported against multiple base tables.Dynamic SQL generation is not supported against a SelectCommand that contains duplicate columns.

Remarks

After the SQL statement is first generated, the application must explicitly call the DbCommandBuilder.RefreshSchema if it changes the ULDataAdapter.SelectCommand in any way. Otherwise, the GetInsertCommand method still uses information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either the DbDataAdapter.Update(System.Data.DataSet) or the GetInsertCommand method.

See also

- "ULCommand class" on page 64
- "SelectCommand property" on page 203
- System.Data.Common.DbCommandBuilder.RefreshSchema

GetInsertCommand(bool) method

Gets the automatically generated ULCommand object required to perform insertions on the database.

Visual Basic syntax

```
Public Shadows Function GetInsertCommand(
    ByVal useColumnsForParameterNames As Boolean
) As ULCommand
```

C# syntax

```
public new ULCommand GetInsertCommand(bool useColumnsForParameterNames)
```

Parameters

- **useColumnsForParameterNames** If true, generate parameter names matching column names if possible. If false, generate @p1, @p2, and so on.

Returns

The automatically generated ULCommand object required to perform insertions.

Exceptions

- **InvalidOperationException** The DbCommandBuilder.DataAdapter has not been initialized.The DataAdapter.SelectCommand property has not been initialized.The DataAdapter.SelectCommand.Connection property has not been initialized.Dynamic SQL generation for the InsertCommand is not supported against a SelectCommand that does not return any modifiable columns.Dynamic SQL generation is not supported against multiple base tables.Dynamic SQL generation is not supported against a SelectCommand that contains duplicate columns.

Remarks

After the SQL statement is first generated, the application must explicitly call the DbCommandBuilder.RefreshSchema if it changes the ULDataAdapter.SelectCommand in any way. Otherwise, the GetInsertCommand method still uses information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either the DbDataAdapter.Update(System.Data.DataSet) or the GetInsertCommand method.

See also

- "ULCommand class" on page 64
- "SelectCommand property" on page 203
- System.Data.Common.DbCommandBuilder.DataAdapter

GetUpdateCommand method

Gets the automatically generated ULCommand object required to perform updates on the database.

Overload list

Name	Description
"GetUpdateCommand() method"	Gets the automatically generated ULCommand object required to perform updates on the database.
"GetUpdateCommand(bool) method"	Gets the automatically generated ULCommand object required to perform updates on the database.

GetUpdateCommand() method

Gets the automatically generated ULCommand object required to perform updates on the database.

Visual Basic syntax

```
Public Shadows Function GetUpdateCommand() As ULCommand
```


C# syntax

```
public new ULCommand GetUpdateCommand()
```

Returns

The automatically generated ULCommand object required to perform updates.

Exceptions

- **InvalidOperationException** The DbCommandBuilder.DataAdapter has not been initialized.The DataAdapter.SelectCommand property has not been initialized.The DataAdapter.SelectCommand.Connection property has not been initialized.Dynamic SQL generation for the UpdateCommand is not supported against a SelectCommand that does not return any modifiable columns.Dynamic SQL generation is not supported against multiple base tables.Dynamic SQL generation is not supported against a SelectCommand that contains duplicate columns.Dynamic SQL generation for the UpdateCommand is not supported against a SelectCommand that does not return any key column information.

Remarks

After the SQL statement is first generated, the application must explicitly call the DbCommandBuilder.RefreshSchema if it changes the ULDataAdapter.SelectCommand in any way. Otherwise, the GetUpdateCommand method still uses information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either the DbDataAdapter.Update(System.Data.DataSet) or the GetUpdateCommand method.

See also

- "ULCommand class" on page 64
- "SelectCommand property" on page 203
- System.Data.Common.DbCommandBuilder.DataAdapter

GetUpdateCommand(bool) method

Gets the automatically generated ULCommand object required to perform updates on the database.

Visual Basic syntax

```
Public Shadows Function GetUpdateCommand(
    ByVal useColumnsForParameterNames As Boolean
) As ULCommand
```

C# syntax

```
public new ULCommand GetUpdateCommand(bool useColumnsForParameterNames)
```

Parameters

- **useColumnsForParameterNames** If true, generate parameter names matching column names if possible. If false, generate @p1, @p2, and so on.

Returns

The automatically generated ULCommand object required to perform updates.

Exceptions

- **InvalidOperationException** The DbCommandBuilder.DataAdapter has not been initialized.The DataAdapter.SelectCommand property has not been initialized.The DataAdapter.SelectCommand.Connection property has not been initialized.Dynamic SQL generation for the UpdateCommand is not supported against a SelectCommand that does not return any modifiable columns.Dynamic SQL generation is not supported against multiple base tables.Dynamic SQL generation is not supported against a SelectCommand that contains duplicate columns.Dynamic SQL generation for the UpdateCommand is not supported against a SelectCommand that does not return any key column information.

Remarks

After the SQL statement is first generated, the application must explicitly call the DbCommandBuilder.RefreshSchema if it changes the ULDataAdapter.SelectCommand in any way. Otherwise, the GetUpdateCommand method still uses information from the previous statement, which might not be correct. The SQL statements are first generated when the application calls either the DbDataAdapter.Update(System.Data.DataSet) or the GetUpdateCommand method.

See also

- "ULCommand class" on page 64
- "SelectCommand property" on page 203
- System.Data.Common.DbCommandBuilder.DataAdapter

DataAdapter property

Gets or sets a ULDataAdapter object for which SQL statements are automatically generated.

Visual Basic syntax

```
Public Shadows Property DataAdapter As ULDataAdapter
```

C# syntax

```
public new ULDataAdapter DataAdapter {get;set;}
```

Remarks

A ULDataAdapter object.

See also

- "ULDataAdapter class" on page 196

ULConnection class

Represents a connection to an UltraLite.NET database.

Visual Basic syntax

```
Public NotInheritable Class ULConnection
    Inherits System.Data.Common.DbConnection
```

C# syntax

```
public sealed class ULConnection : System.Data.Common.DbConnection
```

Base classes

- System.Data.Common.DbConnection

Members

All members of ULConnection class, including all inherited members.

Name	Description
"ULConnection constructor"	Initializes a ULConnection object.
BeginDbTransaction method (Inherited from System.Data.Common.DbConnection)	Starts a database transaction.
"BeginTransaction method"	Returns a transaction object.
"CancelGetNotification method"	**UL Ext:** Cancel any pending get-notification calls on all queues matching the given name.
"ChangeDatabase method"	Changes the current database for an open ULConnection.
"ChangeEncryptionKey method"	**UL Ext:** Changes the database's encryption key to the specified new key.
"ChangePassword method"	Changes the password for the user indicated in the connection string to the supplied new password.
"Close method"	Closes the database connection.
"CountUploadRows method"	**UL Ext:** Returns the number of rows that need to be uploaded when the next synchronization takes place.

Name	Description
"CreateCommand method"	Creates and initializes a ULCommand object associated with this connection and its current transaction.
"CreateNotificationQueue method"	**UL Ext:** Create an event queue.
"DeclareEvent method"	**UL Ext:** Declare a named event.
"DestroyNotificationQueue method"	**UL Ext:** Destroy a event queue.
EnlistTransaction method (Inherited from System.Data.Common.DbConnection)	Enlists in the specified transaction.
"ExecuteTable method"	**UL Ext:** Retrieves in a ULTable a database table for direct manipulation.
"GetLastDownloadTime method"	**UL Ext:** Returns the time of the most recent download of the specified publication.
"GetNewUUID method"	**UL Ext:** Generates a new UUID (System.Guid).
"GetNotification method"	**UL Ext:** Block for notification or timeout.
"GetNotificationParameter method"	**UL Ext:** Get value of a parameter, for an event just read by GetNotification().
"GetSchema method"	Returns the list of supported schema collections.
"GrantConnectTo method"	**UL Ext:** Grants access to an UltraLite database for a user ID with a specified password.
OnStateChange method (Inherited from System.Data.Common.DbConnection)	Raises the System.Data.Common.DbConnection.StateChange event.
"Open method"	Opens a connection to a database using the previously-specified connection string.

Name	Description
“RegisterForEvent method”	**UL Ext:** Registers a queue to get events from an object.
“ResetLastDownloadTime method”	**UL Ext:** Resets the time of the most recent download.
“RevokeConnectFrom method”	**UL Ext:** Revokes access to an UltraLite database from the specified user ID.
“RollbackPartialDownload method”	**UL Ext:** Rolls back outstanding changes to the database from a partial download.
“SendNotification method”	**UL Ext:** Send a notification to matching queues.
“SetSyncListener method”	Specifies the listener object used to process synchronization messages.
“StartSynchronizationDelete method”	**UL Ext:** Marks all subsequent deletes made by this connection for synchronization.
“StopSynchronizationDelete method”	**UL Ext:** Prevents delete operations from being synchronized.
“Synchronize method”	**UL Ext:** Synchronize the database using the current ULConnection.SyncParms.
“TriggerEvent method”	**UL Ext:** Trigger an event.
“ValidateDatabase method”	**UL Ext:** Performs validation on the current database.
“ConnectionString property”	Specifies the parameters to use for opening a connection to an UltraLite.NET database.
“ConnectionTimeout property”	This feature is not supported by UltraLite.NET.

Name	Description
"Database property"	Returns the name of the database to which the connection opens.
"DatabaseID property"	**UL Ext:** Specifies the Database ID value to be used for global autoincrement columns.
"DataSource property"	This feature is not supported by UltraLite.NET.
DbProviderFactory property (Inherited from System.Data.Common.DbConnection)	Gets the System.Data.Common.DbProviderFactory for this System.Data.Common.DbConnection.
"GlobalAutoIncrementUsage property"	**UL Ext:** Returns the percentage of available global autoincrement values that have been used.
"LastIdentity property"	**UL Ext:** Returns the most recent identity value used.
"Schema property"	**UL Ext:** Provides access to the schema of the current database associated with this connection.
"ServerVersion property"	This feature is not supported by UltraLite.NET.
"State property"	Returns the current state of the connection.
"SyncParms property"	**UL Ext:** Specifies the synchronization settings for this connection.
"SyncResult property"	**UL Ext:** Returns the results of the last synchronization for this connection.
"InfoMessage event"	Occurs when UltraLite.NET sends a warning or an informational message on this connection.

Name	Description
“StateChange event”	Occurs when this connection changes state.
“INVALID_DATABASE_ID field”	**UL Ext:** A database ID constant indicating that the ULConnection.DatabaseID has not been set.
“SYNC_ALL_DB field”	Empty publication list, corresponding to the entire database.
“SYNC_ALL_PUBS field”	Publication name "*", corresponding to all publications.

Remarks

To use the UltraLite Engine runtime of UltraLite.NET, set ULDatabaseManager.RuntimeType to the appropriate value before using any other UltraLite.NET API.

A connection to an existing database is opened using the ULConnection.Open.

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. In addition, you must close all result sets and tables opened on a connection before closing the connection.

The schema of the database can be accessed using an open connection's ULConnection.Schema.

See also

- “Open method” on page 136
- “Schema property” on page 149
- System.Data.IDbConnection

ULConnection constructor

Initializes a ULConnection object.

Overload list

Name	Description
“ULConnection() constructor”	Initializes a ULConnection object.
“ULConnection(string) constructor”	Initializes a ULConnection object with the specified connection string.

ULConnection() constructor

Initializes a ULConnection object.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ULConnection()
```

Remarks

The connection must be opened before you can perform any operations against the database.

To use the UltraLite Engine runtime of UltraLite.NET, set ULDatabaseManager.RuntimeType to the appropriate value before using any other UltraLite.NET API.

The ULConnection object needs to have the ULConnection.ConnectionString set before it can be opened.

See also

- "Open method" on page 136
- "ConnectionString property" on page 145

ULConnection(string) constructor

Initializes a ULConnection object with the specified connection string.

Visual Basic syntax

```
Public Sub New(ByVal connectionString As String)
```

C# syntax

```
public ULConnection(string connectionString)
```

Parameters

- **connectionString** An UltraLite.NET connection string. A connection string is a semicolon-separated list of keyword-value pairs.

Exceptions

- **ArgumentException** The supplied connection string is invalid.

Remarks

The connection must be opened before you can perform any operations against the database.

To use the UltraLite Engine runtime of UltraLite.NET, set ULDatabaseManager.RuntimeType to the appropriate value before using any other UltraLite.NET API.

The connection string can be supplied using a ULConnectionParms object.

See also

- "Open method" on page 136
- "ULConnection constructor" on page 115
- "ULConnectionParms class" on page 154
- "ConnectionString property" on page 145

Example

The following code creates and opens a connection to the existing database \UltraLite\MyDatabase.udb on a Windows Mobile device.

```
' Visual Basic
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"
Dim conn As ULConnection = _
    New ULConnection( openParms.ToString() )
conn.Open()
```

The following code is the C# language equivalent:

```
// C#
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnDevice = ".udb";
ULConnection conn = new ULConnection( openParms.ToString() );
conn.Open();
```

BeginTransaction method

Returns a transaction object.

Overload list

Name	Description
"BeginTransaction() method"	Returns a transaction object.
"BeginTransaction(IsolationLevel) method"	Returns a transaction object with the specified isolation level.

BeginTransaction() method

Returns a transaction object.

Visual Basic syntax

```
Public Shadows Function BeginTransaction() As ULTransaction
```

C# syntax

```
public new ULTransaction BeginTransaction()
```

Returns

A ULTransaction object representing the new transaction.

Exceptions

- **"ULException class"** The connection is closed.
- **InvalidOperationException** ULConnection does not support parallel transactions.

Remarks

Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with ULTransaction.Commit or ULTransaction.Rollback.

The transaction is created with IsolationLevel.ReadCommitted.

To associate a command with a transaction object, use the ULCommand.Transaction property. The current transaction is automatically associated to commands created by ULConnection.CreateCommand.

By default, the connection does not use transactions and all commands are automatically committed as they are executed. Once the current transaction is committed or rolled back, the connection reverts to auto commit mode and the previous isolation level until the next call to BeginTransaction.

UltraLite's definition of each isolation level is slightly different than ADO.NET's documentation of IsolationLevel. For more information, see "UltraLite isolation levels" [*UltraLite - Database Management and Reference*].

This is the strongly-typed version of System.Data.IDbConnection.BeginTransaction() and System.Data.Common.DbConnection.BeginTransaction().

See also

- "Commit method" on page 431
- "Rollback method" on page 431
- "Transaction property" on page 98
- "CreateCommand method" on page 123

BeginTransaction(IsolationLevel) method

Returns a transaction object with the specified isolation level.

Visual Basic syntax

```
Public Shadows Function BeginTransaction(
    ByVal isolationLevel As IsolationLevel
) As ULTransaction
```

C# syntax

```
public new ULTransaction BeginTransaction(IsolationLevel isolationLevel)
```


Parameters

- **isolationLevel** The required isolation level for the transaction. UltraLite.NET only supports System.Data.IsolationLevel.ReadUncommitted and ReadCommitted.

Returns

A ULTransaction object representing the new transaction.

Exceptions

- **"ULException class"** The connection is closed or an unsupported isolation level was specified.
- **InvalidOperationException** ULConnection does not support parallel transactions.

Remarks

Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with ULTransaction.Commit or ULTransaction.Rollback.

To associate a command with a transaction object, use the ULCommand.Transaction property. The current transaction is automatically associated to commands created by ULConnection.CreateCommand.

By default, the connection does not use transactions and all commands are automatically committed as they are executed. Once the current transaction is committed or rolled back, the connection reverts to auto commit mode and the previous isolation level until the next call to BeginTransaction.

UltraLite's definition of each isolation level is slightly different than ADO.NET's documentation of IsolationLevel. For more information, see "UltraLite isolation levels" [*UltraLite - Database Management and Reference*].

This is the strongly-typed version of System.Data.IDbConnection.BeginTransaction(System.Data.IsolationLevel) and System.Data.Common.DbConnection.BeginTransaction(System.Data.IsolationLevel).

See also

- "ULTransaction class" on page 430
- "BeginTransaction method" on page 117
- "Commit method" on page 431
- "Rollback method" on page 431
- "Transaction property" on page 98
- "CreateCommand method" on page 123
- System.Data.IsolationLevel

CancelGetNotification method

UL Ext: Cancel any pending get-notification calls on all queues matching the given name.

Visual Basic syntax

```
Public Function CancelGetNotification(
    ByVal queueName As String
) As Integer
```

C# syntax

```
public int CancelGetNotification(string queueName)
```

Parameters

- **queueName** The expression to match queue names upon.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

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

Return the number of affected queues (not the number of blocked reads necessarily).

See also

- "GetNotification method" on page 131
- "ULException class" on page 257

ChangeDatabase method

Changes the current database for an open ULConnection.

Visual Basic syntax

```
Public Overrides Sub ChangeDatabase(ByVal connectionString As String)
```

C# syntax

```
public override void ChangeDatabase(string connectionString)
```

Parameters

- **connectionString** A complete connection string to open the connection to a new database.

Remarks

The connection to the current database is closed even if there are parameter errors.

UL Ext:*connectionString* is a full connection string, not a dbn or dbf.

See also

- "ConnectionString property" on page 145

ChangeEncryptionKey method

UL Ext: Changes the database's encryption key to the specified new key.

Visual Basic syntax

```
Public Sub ChangeEncryptionKey(ByVal newKey As String)
```

C# syntax

```
public void ChangeEncryptionKey(string newKey)
```

Parameters

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

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

If the encryption key is lost, it is not possible to open the database.

See also

- "EncryptionKey property" on page 162

ChangePassword method

Changes the password for the user indicated in the connection string to the supplied new password.

Visual Basic syntax

```
Public Shared Sub ChangePassword(
    ByVal connectionString As String,
    ByVal newPassword As String
)
```

C# syntax

```
public static void ChangePassword(
    string connectionString,
    string newPassword
)
```

Parameters

- **connectionString** The connection string that contains enough information to connect to the database that you want. The connection string may contain the user ID and the current password.
- **newPassword** The new password to set.

Exceptions

- **ArgumentNullException** Either the connectionString or the newPassword parameter is null.
- **ArgumentException** The connection string includes the option to use integrated security.
- **"ULException class"** A SQL error occurred while attempting to open the database.

Close method

Closes the database connection.

Visual Basic syntax

```
Public Overrides Sub Close()
```

C# syntax

```
public override void Close()
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The Close method rolls back any pending transactions and then closes the connection. An application can call Close multiple times.

CountUploadRows method

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

Visual Basic syntax

```
Public Function CountUploadRows(
    ByVal pubs As String,
    ByVal threshold As Long
) As Long
```

C# syntax

```
public long CountUploadRows(string pubs, long threshold)
```

Parameters

- **pubs** A comma separated list of publications to check for rows.
- **threshold** The maximum number of rows to count, limiting the amount of time taken by CountUploadRows. A value of 0 corresponds to the maximum limit. A value of 1 determines if any rows need to be synchronized.

Returns

The number of rows that need to be uploaded from the specified publication(s).

Exceptions

- **"ULException class"** A SQL error occurred.

CreateCommand method

Creates and initializes a ULCommand object associated with this connection and its current transaction.

Visual Basic syntax

```
Public Shadows Function CreateCommand() As ULCommand
```

C# syntax

```
public new ULCommand CreateCommand()
```

Returns

A new ULCommand object.

Remarks

You can use the properties of the ULCommand to control its behavior.

You must set the ULCommand.CommandText before the command can be executed.

This is the strongly-typed version of System.Data.IDbConnection.CreateCommand and System.Data.Common.DbConnection.CreateCommand().

See also

- "ULCommand class" on page 64
- "CommandText property" on page 94
- System.Data.IDbConnection.CreateCommand

CreateNotificationQueue method

UL Ext: Create an event queue.

Visual Basic syntax

```
Public Sub CreateNotificationQueue(
    ByVal queueName As String,
    ByVal parameters As String
)
```

C# syntax

```
public void CreateNotificationQueue(string queueName, string parameters)
```

Parameters

- **queueName** The name of the new queue.
- **parameters** Creation parameters; currently unused, set to NULL.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Create an event notification queue for this connection. Queue names are scoped per-connection, so different connections can create queues with the same name. When an event notification is sent, all queues in the database with a matching name receive (a separate instance of) the notification. Names are case insensitive. A default queue is created on demand for each connection when calling RegisterForEvent() if no queue is specified.

See also

- "DestroyNotificationQueue method" on page 125
- "ULException class" on page 257

DeclareEvent method

UL Ext: Declare a named event.

Visual Basic syntax

```
Public Sub DeclareEvent(ByVal eventName As String)
```

C# syntax

```
public void DeclareEvent(string eventName)
```

Parameters

- **eventName** The event name.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Declare an event which can then be registered for and triggered. UltraLite predefines some system events triggered by operations on the database or the environment. The event name must be unique. Names are case insensitive. Throws error if name already used or invalid

See also

- "CreateNotificationQueue method" on page 123
- "ULException class" on page 257

DestroyNotificationQueue method

UL Ext: Destroy a event queue.

Visual Basic syntax

```
Public Sub DestroyNotificationQueue(ByVal queueName As String)
```

C# syntax

```
public void DestroyNotificationQueue(string queueName)
```

Parameters

- **queueName** The name of the queue.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Destroy the given event notification queue. A warning is signaled if unread notifications remain in the queue. Unread notifications are discarded. A connection's default event queue, if created, is destroyed when the connection is closed.

See also

- "CreateNotificationQueue method" on page 123
- "ULException class" on page 257

ExecuteTable method

UL Ext: Retrieves in a ULTable a database table for direct manipulation.

Overload list

Name	Description
"ExecuteTable(string) method"	**UL Ext:** Retrieves in a ULTable a database table for direct manipulation.
"ExecuteTable(string, string) method"	**UL Ext:** Retrieves in a ULTable a database table for direct manipulation.
"ExecuteTable(string, string, CommandBehavior) method"	**UL Ext:** Retrieves, with the specified command behavior, a database table for direct manipulation.

ExecuteTable(string) method

UL Ext: Retrieves in a ULTable a database table for direct manipulation.

Visual Basic syntax

```
Public Function ExecuteTable(ByVal tableName As String) As ULTable
```

C# syntax

```
public ULTable ExecuteTable(string tableName)
```

Parameters

- **tableName** The name of the table to open.

Returns

The table as a ULTable object.

Exceptions

- **"ULException class"** A SQL error occurred.
- **InvalidOperationException** The *tableName* is invalid.

Remarks

The table is opened (sorted) using the table's primary key.

This method is a shortcut for the ULCommand.ExecuteTable() method that does not require a ULCommand instance. It is provided to help users porting from earlier versions of UltraLite.NET (it replaces iAnywhere.UltraLite.Connection.GetTable() and iAnywhere.UltraLite.Table.Open()).

See also

- "ExecuteTable method" on page 125
- "ULTable class" on page 394
- "ExecuteTable method" on page 91
- "ULCommand class" on page 64

Example

The following code opens the table named MyTable using the table's primary key. It assumes an open ULConnection instance called conn.

```
' Visual Basic
Dim t As ULTable = conn.ExecuteTable("MyTable")

' The line above is equivalent to
' Dim cmd As ULCommand = conn.CreateCommand()
' cmd.CommandText = "MyTable"
' cmd.CommandType = CommandType.TableDirect
' Dim t As ULTable = cmd.ExecuteTable()
' cmd.Dispose()
```

The following code is the C# language equivalent:


```
// C#
ULTable t = conn.ExecuteTable("MyTable");

// The line above is equivalent to
// ULTable t;
// using(ULCommand cmd = conn.CreateCommand())
// {
//     cmd.CommandText = "MyTable";
//     cmd.CommandType = CommandType.TableDirect;
//     t = cmd.ExecuteTable();
// }
```

ExecuteTable(string, string) method

UL Ext: Retrieves in a ULTable a database table for direct manipulation.

Visual Basic syntax

```
Public Function ExecuteTable(
    ByVal tableName As String,
    ByVal indexName As String
) As ULTable
```

C# syntax

```
public ULTable ExecuteTable(string tableName, string indexName)
```

Parameters

- **tableName** The name of the table to open.
- **indexName** The name of the index with which to open (sort) the table.

Returns

The table as a ULTable object.

Exceptions

- **"ULException class"** A SQL error occurred.
- **InvalidOperationException** The *tableName* is invalid.

Remarks

The table is opened (sorted) using the specified index.

This method is a shortcut for the ULCommand.ExecuteTable() method that does not require a ULCommand instance. It is provided to help users porting from earlier versions of UltraLite.NET (it replaces iAnywhere.UltraLite.Connection.GetTable() and iAnywhere.UltraLite.Table.Open()).

See also

- "ExecuteTable method" on page 125
- "ULTable class" on page 394
- "ExecuteTable method" on page 91
- "ULCommand class" on page 64

Example

The following code opens the table named MyTable using the index named MyIndex. It assumes an open ULConnection instance called conn.

```
' Visual Basic
Dim t As ULTable = conn.ExecuteTable("MyTable", "MyIndex")

' The line above is equivalent to
' Dim cmd As ULCommand = conn.CreateCommand()
' cmd.CommandText = "MyTable"
' cmd.IndexName = "MyIndex"
' cmd.CommandType = CommandType.TableDirect
' Dim t As ULTable = cmd.ExecuteTable()
' cmd.Dispose()
```

The following code is the C# language equivalent:

```
// C#
ULTable t = conn.ExecuteTable("MyTable", "MyIndex");

// The line above is equivalent to
// ULTable t;
// using(ULCommand cmd = conn.CreateCommand())
// {
//     cmd.CommandText = "MyTable";
//     cmd.IndexName = "MyIndex";
//     cmd.CommandType = CommandType.TableDirect;
//     t = cmd.ExecuteTable();
// }
```

ExecuteTable(string, string, CommandBehavior) method

UL Ext: Retrieves, with the specified command behavior, a database table for direct manipulation.

Visual Basic syntax

```
Public Function ExecuteTable(
    ByVal tableName As String,
    ByVal indexName As String,
    ByVal cmdBehavior As CommandBehavior
) As ULTable
```

C# syntax

```
public ULTable ExecuteTable(
    string tableName,
    string indexName,
    CommandBehavior cmdBehavior
)
```


Parameters

- **tableName** The name of the table to open.
- **indexName** The name of the index with which to open (sort) the table.
- **cmdBehavior** A bitwise combination of System.Data.CommandBehavior flags describing the results of the query and its effect on the connection. UltraLite.NET respects only the System.Data.CommandBehavior.Default, System.Data.CommandBehavior.CloseConnection, and System.Data.CommandBehavior.SchemaOnly flags.

Returns

The table as a ULTable object.

Exceptions

- **"ULException class"** A SQL error occurred.
- **InvalidOperationException** The *tableName* is invalid.

Remarks

The table is opened (sorted) using the specified index.

This method is a shortcut for the ULCommand.ExecuteTable(System.Data.CommandBehavior) that does not require a ULCommand instance. It is provided to help users porting from earlier versions of UltraLite.NET (it replaces iAnywhere.UltraLite.Connection.GetTable() and iAnywhere.UltraLite.Table.Open()).

See also

- "ExecuteTable method" on page 125
- "ULTable class" on page 394
- "ULCommand class" on page 64
- System.Data.CommandBehavior

Example

The following code opens the table named MyTable using the index named MyIndex. It assumes an open ULConnection instance called conn.

```
' Visual Basic
Dim t As ULTable = conn.ExecuteTable( _
    "MyTable", "MyIndex", CommandBehavior.Default _
)

' The line above is equivalent to the following code:
' Dim cmd As ULCommand = conn.CreateCommand()
' cmd.CommandText = "MyTable"
' cmd.IndexName = "MyIndex"
' cmd.CommandType = CommandType.TableDirect
' Dim t As ULTable = cmd.ExecuteTable(CommandBehavior.Default)
' cmd.Dispose()
```

The following code is the C# language equivalent:

```
// C#
ULTable t = conn.ExecuteTable(
    "MyTable", "MyIndex", CommandBehavior.Default
);

// The line above is equivalent to the following code:
// ULTable t;
// using(ULCommand cmd = conn.CreateCommand())
// {
//     cmd.CommandText = "MyTable";
//     cmd.IndexName = "MyIndex";
//     cmd.CommandType = CommandType.TableDirect;
//     t = cmd.ExecuteTable(CommandBehavior.Default);
// }
```

GetLastDownloadTime method

UL Ext: Returns the time of the most recent download of the specified publication.

Visual Basic syntax

```
Public Function GetLastDownloadTime(ByVal publication As String) As Date
```

C# syntax

```
public DateTime GetLastDownloadTime(string publication)
```

Parameters

- **publication** The publication to check.

Returns

The timestamp of the last download.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The parameter *publication* is a publication name to check. If the special constant SYNC_ALL_DB is used, returns the time of the last download of the entire database.

See also

- "ResetLastDownloadTime method" on page 137
- "SYNC_ALL_DB field" on page 154

GetNewUUID method

UL Ext: Generates a new UUID (System.Guid).

Visual Basic syntax

```
Public Function GetNewUUID() As Guid
```

C# syntax

```
public Guid GetNewUUID()
```

Returns

A new UUID as a System.Guid.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

This method is provided here because it is not included in the .NET Compact Framework.

See also

- System.Guid

GetNotification method

UL Ext: Block for notification or timeout.

Visual Basic syntax

```
Public Function GetNotification(
    ByVal queueName As String,
    ByVal wait_ms As Integer
) As String
```

C# syntax

```
public string GetNotification(string queueName, int wait_ms)
```

Parameters

- **queueName** The name of the queue to be waited upon.
- **wait_ms** The time to wait, in milliseconds. Use System.Threading.Timeout.Infinite (-1) for an indefinite wait.

Returns

Return null if wait period expired or was canceled; otherwise, returns the event name.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Read an event notification. This call blocks until a notification is received or until the given wait period expires. To wait indefinitely, pass System.Threading.Timeout.Infinite for \p wait_ms. To cancel a wait, send another notification to the given queue or use CancelGetNotification(). After reading a notification, use ReadNotificationParameter() to retrieve additional parameters.

See also

- "SendNotification method" on page 139
- "GetNotificationParameter method" on page 132
- "CancelGetNotification method" on page 119
- "ULException class" on page 257
- System.Threading.Timeout

GetNotificationParameter method

UL Ext: Get value of a parameter, for an event just read by GetNotification().

Visual Basic syntax

```
Public Function GetNotificationParameter(
    ByVal queueName As String,
    ByVal parameterName As String
) As String
```

C# syntax

```
public string GetNotificationParameter(
    string queueName,
    string parameterName
)
```

Parameters

- **queueName** The name of the queue to be waited upon.
- **parameterName** The name of the parameter whose value should be returned.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Get a parameter for the event notification just read by ULGetNotification(). Only the parameters from the most-recently read notification on the given queue are available.

Returns parameter value if the parameter was found; otherwise, returns null.

See also

- "GetNotification method" on page 131
- "ULException class" on page 257

GetSchema method

Returns the list of supported schema collections.

Overload list

Name	Description
"GetSchema() method"	Returns the list of supported schema collections.
"GetSchema(string) method"	Returns information for the specified metadata collection for this ULConnection.
"GetSchema(string, string[]) method"	Returns schema information for the data source of this ULConnection and, if specified, uses the specified string for the schema name and the specified string array for the restriction values.

GetSchema() method

Returns the list of supported schema collections.

Visual Basic syntax

```
Public Overrides Function GetSchema() As DataTable
```

C# syntax

```
public override DataTable GetSchema()
```

GetSchema(string) method

Returns information for the specified metadata collection for this ULConnection.

Visual Basic syntax

```
Public Overrides Function GetSchema(
    ByVal collection As String
) As DataTable
```

C# syntax

```
public override DataTable GetSchema(string collection)
```

Parameters

- **collection** Name of the metadata collection. If none provided, MetaDataCollections is used.

See also

- "ULConnection class" on page 110

GetSchema(string, string[]) method

Returns schema information for the data source of this ULConnection and, if specified, uses the specified string for the schema name and the specified string array for the restriction values.

Visual Basic syntax

```
Public Overrides Function GetSchema(
    ByVal collection As String,
    ByVal restrictions As String()
) As DataTable
```

C# syntax

```
public override DataTable GetSchema(
    string collection,
    string[] restrictions
)
```

Parameters

- **collection** Name of the metadata collection. If none provided, MetaDataCollections is used.
- **restrictions** A set of restriction values for the requested schema.

Returns

A DataTable that contains schema information.

Remarks

This method is used to query the database for various metadata. Each type of metadata is given a collection name, which must be passed to receive that data. The default collection name is MetaDataCollections.

You can query the .NET data provider to determine the list of supported schema collections by calling the GetSchema method with no arguments, or with the schema collection name MetaDataCollections. This returns a DataTable with a list of the supported schema collections (CollectionName), the number of restrictions that they each support (NumberOfRestrictions), and the number of identifier parts that they use (NumberOfIdentifierParts).

Collection	Metadata
Columns	Returns information about all columns in the database.
DataSourceInformation	Returns information about the database provider.

Collection	Metadata
DataTypes	Returns a list of supported data types.
ForeignKeys	Returns information about all foreign keys in the database.
IndexColumns	Returns information about all index columns in the database.
Indexes	Returns information about all indexes in the database.
MetaDataCollections	Returns a list of all collection names.
Publications	Returns information about all publications in the database.
ReservedWords	Returns a list of reserved words used by UltraLite.
Restrictions	Returns information about restrictions used in GetSchema.
Tables	Returns information about all tables in the database.

These collection names are also available as read-only properties in the ULMetaDataCollectionNames class.

The results returned can be filtered by specifying an array of restrictions in the call to GetSchema.

The restrictions available with each collection can be queried by calling:

```
GetSchema( "Restrictions" )
```

If the collection requires four restrictions, then the restrictions parameter must be either NULL, or a string with four values.

To filter on a particular restriction, place the string to filter by in its place in the array and leave any unused places NULL. For example, the Tables collection has three restrictions: Table, TableType, SyncType.

To filter the Table collection:

GetSchema("Tables", new string[ ] { "my_table", NULL, NULL }) Returns information about all tables named my_table.

GetSchema("Tables", new string[ ] { NULL, "User", NULL }) Returns information about all user tables.

See also

- “ULConnection class” on page 110
- “ULMetaDataCollectionNames class” on page 293

GrantConnectTo method

UL Ext: Grants access to an UltraLite database for a user ID with a specified password.

Visual Basic syntax

```
Public Sub GrantConnectTo(ByVal uid As String, ByVal pwd As String)
```

C# syntax

```
public void GrantConnectTo(string uid, string pwd)
```

Parameters

- **uid** The user ID to receive access to the database. The maximum length of the user ID is 16 characters.
- **pwd** The password to be associated with the user ID. The maximum length is 16 characters.

Remarks

If an existing user ID is specified, this function updates the password for the user. UltraLite supports a maximum of 4 users. This method is enabled only if user authentication was enabled when the connection was opened.

See also

- "UserID property" on page 163
- "Password property" on page 162
- "ConnectionString property" on page 145

Open method

Opens a connection to a database using the previously-specified connection string.

Visual Basic syntax

```
Public Overrides Sub Open()
```

C# syntax

```
public override void Open()
```

Exceptions

- **InvalidOperationException** The connection is already open or the connection string is not specified in ULConnection.ConnectionString.
- **"ULException class"** A SQL error occurred while attempting to open the database.

Remarks

You should explicitly close or dispose of the connection when you are done with it.

See also

- "ConnectionString property" on page 145
- "State property" on page 150

RegisterForEvent method

UL Ext: Registers a queue to get events from an object.

Visual Basic syntax

```
Public Sub RegisterForEvent(
    ByVal eventName As String,
    ByVal objectName As String,
    ByVal queueName As String,
    ByVal registerNotUnReg As Boolean
)
```

C# syntax

```
public void RegisterForEvent(
    string eventName,
    string objectName,
    string queueName,
    bool registerNotUnReg
)
```

Parameters

- **eventName** The event name.
- **objectName** The object name to which event applies. For example, a table name.
- **queueName** The event queue name to be used.
- **registerNotUnReg** True to register; false to unregister.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

This method registers a queue to receive notifications of an event. The default connection queue is implied and created if a queue name is not supplied. Certain system events allow specification of an object name to which the event applies. For example, the TableModified event can specify the table name. Unlike SendNotification(), only the specific queue registered receives notifications of the event; other queues with the same name on different connections do not (unless they are also explicit registered). This method throws an error if the queue or event does not exist.

See also

- "DeclareEvent method" on page 124
- "CreateNotificationQueue method" on page 123
- "ULException class" on page 257

ResetLastDownloadTime method

UL Ext: Resets the time of the most recent download.

Visual Basic syntax

```
Public Sub ResetLastDownloadTime(ByVal pubs As String)
```

C# syntax

```
public void ResetLastDownloadTime(string pubs)
```

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetLastDownloadTime method" on page 130

RevokeConnectFrom method

UL Ext: Revokes access to an UltraLite database from the specified user ID.

Visual Basic syntax

```
Public Sub RevokeConnectFrom(ByVal uid As String)
```

C# syntax

```
public void RevokeConnectFrom(string uid)
```

Parameters

- **uid** The user ID whose access to the database is being revoked.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GrantConnectTo method" on page 135

RollbackPartialDownload method

UL Ext: Rolls back outstanding changes to the database from a partial download.

Visual Basic syntax

```
Public Sub RollbackPartialDownload()
```

C# syntax

```
public void RollbackPartialDownload()
```


Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "KeepPartialDownload property" on page 375
- "ResumePartialDownload property" on page 378

SendNotification method

UL Ext: Send a notification to matching queues.

Visual Basic syntax

```
Public Function SendNotification(
    ByVal queueName As String,
    ByVal eventName As String,
    ByVal parameters As String
) As Integer
```

C# syntax

```
public int SendNotification(
    string queueName,
    string eventName,
    string parameters
)
```

Parameters

- **queueName** The event queue name to be used.
- **eventName** The event name.
- **parameters** Parameters to pass.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Returns the number of matching queues.

Send a notification to all queues matching the given name (including any such queue on the current connection). This call does not block. Use the special queue name "*" to send to all queues.

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

See also

- "DeclareEvent method" on page 124
- "RegisterForEvent method" on page 137
- "ULException class" on page 257

SetSyncListener method

Specifies the listener object used to process synchronization messages.

Visual Basic syntax

```
Public Sub SetSyncListener(ByVal listener As ULSyncProgressListener)
```

C# syntax

```
public void SetSyncListener(ULSyncProgressListener listener)
```

Parameters

- **listener** The ULSyncProgressListener object that implements SyncProgressed(), which is called for synchronization messages on this connection.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

When the SYNCHRONIZE profileName SQL statement is executed, its progress messages are routed to syncListener, if not null (Nothing in Visual Basic).

To remove the listener, pass a null reference in a call to SetSyncListener.

See also

- "ULSyncProgressListener interface" on page 389

StartSynchronizationDelete method

UL Ext: Marks all subsequent deletes made by this connection for synchronization.

Visual Basic syntax

```
Public Sub StartSynchronizationDelete()
```

C# syntax

```
public void StartSynchronizationDelete()
```


Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

When this function is called, all delete operations are again synchronized, causing the rows deleted from the UltraLite database to be removed from the consolidated database as well.

See also

- "StopSynchronizationDelete method" on page 141
- "Truncate method" on page 416

StopSynchronizationDelete method

UL Ext: Prevents delete operations from being synchronized.

Visual Basic syntax

```
Public Sub StopSynchronizationDelete()
```

C# syntax

```
public void StopSynchronizationDelete()
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

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

See also

- "StartSynchronizationDelete method" on page 140

Synchronize method

UL Ext: Synchronize the database using the current ULConnection.SyncParms.

Overload list

Name	Description
"Synchronize() method"	**UL Ext:** Synchronize the database using the current ULConnection.SyncParms.

Name	Description
"Synchronize(ULSyncProgressListener) method"	**UL Ext:** Synchronizes the database using the current ULConnection.SyncParms with progress events posted to the specified listener.

Synchronize() method

UL Ext: Synchronize the database using the current ULConnection.SyncParms.

Visual Basic syntax

```
Public Sub Synchronize()
```

C# syntax

```
public void Synchronize()
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

A detailed result status is reported in this connection's ULConnection.SyncResult.

See also

- "Synchronize method" on page 141
- "SyncParms property" on page 151
- "SyncResult property" on page 151

Synchronize(ULSyncProgressListener) method

UL Ext: Synchronizes the database using the current ULConnection.SyncParms with progress events posted to the specified listener.

Visual Basic syntax

```
Public Sub Synchronize(ByVal listener As ULSyncProgressListener)
```

C# syntax

```
public void Synchronize(ULSyncProgressListener listener)
```

Parameters

- **listener** The object that receives synchronization progress events.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Errors during synchronization are posted as a ULSyncProgressState.STATE_ERROR event, then thrown as a ULException.

A detailed result status is reported in this connection's ULConnection.SyncResult.

See also

- "ULSyncProgressListener interface" on page 389
- "Synchronize method" on page 141
- "SyncParms property" on page 151
- "ULException class" on page 257
- "SyncResult property" on page 151

TriggerEvent method

UL Ext: Trigger an event.

Visual Basic syntax

```
Public Function TriggerEvent(
    ByVal eventName As String,
    ByVal parameters As String
) As Integer
```

C# syntax

```
public int TriggerEvent(string eventName, string parameters)
```

Parameters

- **eventName** The event name to be triggered.
- **parameters** Parameters to pass.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Returns the number of notifications sent.

Trigger an event (and send notification to all registered queues).

Returns the number of event notifications sent.

See also

- "DeclareEvent method" on page 124
- "RegisterForEvent method" on page 137
- "ULException class" on page 257

ValidateDatabase method

UL Ext: Performs validation on the current database.

Overload list

Name	Description
"ValidateDatabase(ULDBValid) method"	**UL Ext:** Performs validation on the current database.
"ValidateDatabase(ULDBValid, string) method"	**UL Ext:** Performs validation on the current database.

ValidateDatabase(ULDBValid) method

UL Ext: Performs validation on the current database.

Visual Basic syntax

```
Public Sub ValidateDatabase(ByVal how As ULDBValid)
```

C# syntax

```
public void ValidateDatabase(ULDBValid how)
```

Parameters

- **how** Describes how to validate the database.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ValidateDatabase method" on page 212
- "ULDBValid enumeration" on page 437

Example

The following code validates the current database

```
' Visual Basic
conn.ValidateDatabase( iAnywhere.Data.UltraLite.ULVF_INDEX )
```

The following code is the C# language equivalent:

```
// C#
conn.ValidateDatabase( iAnywhere.Data.UltraLite.ULVF_INDEX )
```

ValidateDatabase(ULDBValid, string) method

UL Ext: Performs validation on the current database.

Visual Basic syntax

```
Public Sub ValidateDatabase(
    ByVal how As ULDBValid,
    ByVal tableName As String
)
```

C# syntax

```
public void ValidateDatabase(ULDBValid how, string tableName)
```

Parameters

- **how** Describes how to validate the database.
- **tableName** If null (Nothing in Visual Basic), validate the entire database; otherwise, validate just the named table.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ValidateDatabase method" on page 212
- "ULDBValid enumeration" on page 437

Example

The following code validates the current database

```
' Visual Basic
conn.ValidateDatabase( iAnywhere.Data.UltraLite.ULVF_INDEX, Nothing )
```

The following code is the C# language equivalent:

```
// C#
conn.ValidateDatabase( iAnywhere.Data.UltraLite.ULVF_INDEX, null )
```

ConnectionString property

Specifies the parameters to use for opening a connection to an UltraLite.NET database.

Visual Basic syntax

```
Public Overrides Property ConnectionString As String
```

C# syntax

```
public override string ConnectionString {get;set;}
```

Exceptions

- **InvalidOperationException** The value cannot be set while the connection is open.
- **ArgumentException** The supplied connection string is invalid.

Remarks

The connection string can be supplied using a ULConnectionParms object.

The parameters used to open this connection in the form of a semicolon-separated list of keyword-value pairs. The default is an empty string (an invalid connection string).

UL Ext: The parameters used by UltraLite.NET are specific to UltraLite databases and therefore the connection string is not compatible with SQL Anywhere connection strings. For a list of parameters, see "UltraLite connection parameters" [*UltraLite - Database Management and Reference*].

Parameter values can be quoted with either single quote characters or double quote characters provided that the quoted contents do not contain quote characters of the same type. Values must be quoted if they contain semicolons, begin with a quote, or require leading or trailing whitespace.

If you are not quoting parameter values, make sure that they do not contain semicolons, and that they begin with either a single quote or a double quote character. Leading and trailing spaces in values are ignored.

By default, connections are opened with UID=DBA and PWD=sql. To make the database more secure, change the user DBA's password or create new users (using GrantConnectTo) and remove the DBA user (using RevokeConnectFrom).

See also

- "Open method" on page 136
- "ULConnectionParms class" on page 154
- "GrantConnectTo method" on page 135

Example

The following code creates and opens a connection to the existing database \UltraLite\MyDatabase.udb on a Windows Mobile device.

```
' Visual Basic
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"
Dim conn As ULConnection = New ULConnection
conn.ConnectionString = openParms.ToString()
conn.Open()
```

The following code is the C# language equivalent:

```
// C#
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnDevice = ".udb";
ULConnection conn = new ULConnection();
conn.ConnectionString = openParms.ToString();
conn.Open();
```

ConnectionTimeout property

This feature is not supported by UltraLite.NET.

Visual Basic syntax

```
Public ReadOnly Overrides Property ConnectionTimeout As Integer
```

C# syntax

```
public override int ConnectionTimeout {get;}
```

Exceptions

- **"ULException class"** Setting the value is not supported in UltraLite.NET.

Remarks

The value is always zero.

Database property

Returns the name of the database to which the connection opens.

Visual Basic syntax

```
Public ReadOnly Overrides Property Database As String
```

C# syntax

```
public override string Database {get;}
```

Remarks

A string containing the name of the database.

On Windows Mobile devices, ULConnection looks in the connection string in the following order: dbn, ce_file.

On desktop machines, ULConnection looks in the connection string in the following order: dbn, nt_file.

DatabaseID property

UL Ext: Specifies the Database ID value to be used for global autoincrement columns.

Visual Basic syntax

```
Public Property DatabaseID As Long
```

C# syntax

```
public long DatabaseID {get;set;}
```

Exceptions

- **"ULException class"** The specified new database ID is invalid.

Remarks

The Database ID value of the current database.

The database ID value must be in the range [0,System.UInt32.MaxValue]. A value of ULConnection.INVALID_DATABASE_ID is used to indicate that the database ID has not been set for the current database.

See also

- "GetDatabaseProperty method" on page 214
- "SetDatabaseOption method" on page 217
- "INVALID_DATABASE_ID field" on page 153
- System.UInt32.MaxValue

DataSource property

This feature is not supported by UltraLite.NET.

Visual Basic syntax

```
Public ReadOnly Overrides Property DataSource As String
```

C# syntax

```
public override string DataSource {get;}
```

Remarks

The value is always the empty string.

GlobalAutoIncrementUsage property

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

Visual Basic syntax

```
Public ReadOnly Property GlobalAutoIncrementUsage As Short
```

C# syntax

```
public short GlobalAutoIncrementUsage {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The percentage of available global autoincrement values that have been used. It is an integer in the range [0-100], inclusive.

If the percentage approaches 100, your application should set a new value for the global database ID using ULConnection.DatabaseID.

See also

- "ULDatabaseManager class" on page 206
- "DatabaseID property" on page 147

LastIdentity property

UL Ext: Returns the most recent identity value used.

Visual Basic syntax

```
Public ReadOnly Property LastIdentity As ULong
```

C# syntax

```
public ulong LastIdentity {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The most recently-used identity value as an unsigned long.

The most recent identity value used. This property is equivalent to the SQL Anywhere statement:

```
SELECT @identity
```

LastIdentity is particularly useful in the context of global autoincrement columns.

Since this property only allows you to determine the most recently assigned default value, you should retrieve this value soon after executing the insert statement to avoid spurious results.

Occasionally, a single insert statement may include more than one column of type global autoincrement. In this case, LastIdentity is one of the generated default values, but there is no reliable means to determine from which column the value is. For this reason, you should design your database and write your insert statements to avoid this situation.

Schema property

UL Ext: Provides access to the schema of the current database associated with this connection.

Visual Basic syntax

```
Public ReadOnly Property Schema As ULDatabaseSchema
```

C# syntax

```
public ULDatabaseSchema Schema {get;}
```

Remarks

A reference to the ULDatabaseSchema object representing the schema of the database on which this connection opens.

This property is only valid while its connection is open.

See also

- "ULDatabaseSchema class" on page 213

ServerVersion property

This feature is not supported by UltraLite.NET.

Visual Basic syntax

```
Public ReadOnly Overrides Property ServerVersion As String
```

C# syntax

```
public override string ServerVersion {get;}
```

Remarks

The value is always the empty string.

State property

Returns the current state of the connection.

Visual Basic syntax

```
Public ReadOnly Overrides Property State As ConnectionState
```

C# syntax

```
public override ConnectionState State {get;}
```

Remarks

Returns System.Data.ConnectionState.Open if the connection is open, or System.Data.ConnectionState.Closed if the connection is closed.

See also

- "StateChange event" on page 152
- System.Data.ConnectionState

SyncParms property

UL Ext: Specifies the synchronization settings for this connection.

Visual Basic syntax

```
Public ReadOnly Property SyncParms As ULSyncParms
```

C# syntax

```
public ULSyncParms SyncParms {get;}
```

Remarks

A reference to the ULSyncParms object representing the parameters used for synchronization by this connection. Modifications to the parameters affect the next synchronization made over this connection.

See also

- "Synchronize method" on page 141
- "SyncResult property" on page 151
- "ULSyncParms class" on page 371

SyncResult property

UL Ext: Returns the results of the last synchronization for this connection.

Visual Basic syntax

```
Public ReadOnly Property SyncResult As ULSyncResult
```

C# syntax

```
public ULSyncResult SyncResult {get;}
```

Remarks

A reference to the ULSyncResult object representing the results of the last synchronization for this connection.

See also

- "Synchronize method" on page 141
- "SyncParms property" on page 151
- "SyncResult property" on page 151

InfoMessage event

Occurs when UltraLite.NET sends a warning or an informational message on this connection.

Visual Basic syntax

```
Public Event InfoMessage As ULInfoMessageEventHandler
```

C# syntax

```
public event ULInfoMessageEventHandler InfoMessage;
```

Remarks

To process UltraLite.NET warnings or informational messages, you must create a ULInfoMessageEventHandler delegate and attach it to this event.

See also

- "ULInfoMessageEventHandler delegate" on page 433

Example

The following code defines an informational message event handler:

```
' Visual Basic
Private Sub MyInfoMessageHandler( _
    obj As Object, args As ULInfoMessageEventArgs _
)
    System.Console.WriteLine( _
        "InfoMesageHandler: " + args.NativeError + ", " _
        + args.Message _
    )
End Sub
```

The following code is the C# language equivalent:

```
// C#
private void MyInfoMessageHandler(
    object obj, ULInfoMessageEventArgs args
)
{
    System.Console.WriteLine(
        "InfoMesageHandler: " + args.NativeError + ", "
        + args.Message
    );
}
```

The following code adds the MyInfoMessageHandler to the connection named conn.

```
' Visual Basic
AddHandler conn.InfoMessage, AddressOf MyInfoMessageHandler
```

The following code is the C# language equivalent:

```
// C#
conn.InfoMessage +=
    new ULInfoMessageEventHandler(MyInfoMessageHandler);
```

StateChange event

Occurs when this connection changes state.

Visual Basic syntax

```
Public Event StateChange As StateChangeEventHandler
```

C# syntax

```
public event override StateChangeEventHandler StateChange;
```

Remarks

To process state change messages, you must create a System.Data.StateChangeEventHandler delegate and attach it to this event.

See also

- System.Data.StateChangeEventHandler

Example

The following code defines a state change event handler.

```
' Visual Basic
Private Sub MyStateHandler( _
    obj As Object, args As StateChangeEventArgs _
)
    System.Console.WriteLine( _
        "StateHandler: " + args.OriginalState + " to " _
        + args.CurrentState _
    )
End Sub
```

The following code is the C# language equivalent:

```
// C#
private void MyStateHandler(
    object obj, StateChangeEventArgs args
)
{
    System.Console.WriteLine(
        "StateHandler: " + args.OriginalState + " to "
        + args.CurrentState
    );
}
```

The following code adds the MyStateHandler to the connection named conn.

```
' Visual Basic
AddHandler conn.StateChange, AddressOf MyStateHandler
```

The following code is the C# language equivalent:

```
// C#
conn.StateChange += new StateChangeEventHandler(MyStateHandler);
```

INVALID_DATABASE_ID field

UL Ext: A database ID constant indicating that the ULConnection.DatabaseID has not been set.

Visual Basic syntax

```
Public Const INVALID_DATABASE_ID As Long
```

C# syntax

```
public const long INVALID_DATABASE_ID;
```

See also

- "DatabaseID property" on page 147

SYNC_ALL_DB field

Empty publication list, corresponding to the entire database.

Visual Basic syntax

```
Public Const SYNC_ALL_DB As String
```

C# syntax

```
public const String SYNC_ALL_DB;
```

SYNC_ALL_PUBS field

Publication name "*", corresponding to all publications.

Visual Basic syntax

```
Public Const SYNC_ALL_PUBS As String
```

C# syntax

```
public const String SYNC_ALL_PUBS;
```

ULConnectionParms class

UL Ext: Builds a connection string for opening a connection to an UltraLite database.

Visual Basic syntax

```
Public Class ULConnectionParms Inherits System.ComponentModel.Component
```

C# syntax

```
public class ULConnectionParms : System.ComponentModel.Component
```

Base classes

- System.ComponentModel.Component

Members

All members of ULConnectionParms class, including all inherited members.

Name	Description
"ULConnectionParms constructor"	Initializes a ULConnectionParms instance with its default values.
Dispose method (Inherited from System.ComponentModel.Component)	Releases all resources used by the System.ComponentModel.Component.
Finalize method (Inherited from System.ComponentModel.Component)	Releases unmanaged resources and performs other cleanup operations before the System.ComponentModel.Component is reclaimed by garbage collection.
GetService method (Inherited from System.ComponentModel.Component)	Returns an object that represents a service provided by the System.ComponentModel.Component or by its System.ComponentModel.Container.
"ToString method"	Returns the string representation of this instance.
"AdditionalParms property"	Specifies additional parameters as a semicolon-separated list of name=value pairs.
"CacheSize property"	Specifies the size of the cache.
CanRaiseEvents property (Inherited from System.ComponentModel.Component)	Gets a value indicating whether the component can raise an event.
"ConnectionName property"	Specifies a name for the connection.
Container property (Inherited from System.ComponentModel.Component)	Gets the System.ComponentModel.IContainer that contains the System.ComponentModel.Component.
"DatabaseOnDesktop property"	Specifies the path and file name of the UltraLite database on Windows desktop platforms.
"DatabaseOnDevice property"	Specifies the path and file name of the UltraLite database on Windows Mobile.
DesignMode property (Inherited from System.ComponentModel.Component)	Gets a value that indicates whether the System.ComponentModel.Component is currently in design mode.
"EncryptionKey property"	Specifies a key for encrypting the database.

Name	Description
Disposed (Inherited from System.ComponentModel.Component) Gets the list of event handlers that are attached to this System.ComponentModel.Component.Events property (Inherited from System.ComponentModel.Component)	Occurs when the component is disposed by a call to the System.ComponentModel.Component.Dispose method.
"Password property"	Specifies the password for the authenticated user.
Site property (Inherited from System.ComponentModel.Component)	Gets or sets the System.ComponentModel.ISite of the System.ComponentModel.Component.
"UserID property"	Specifies an authenticated user for the database.

Remarks

The frequently-used connection parameters are individual properties on the ULConnectionParms object.

A ULConnectionParms object is used to specify the parameters for opening a connection (ULConnection.Open) or dropping a database (ULDatabaseManager.DropDatabase).

Leading and trailing spaces are ignored in all values. Values must not contain leading or trailing spaces, or a semicolon, or begin with either a single quote or a double quote.

When building a connection string, you need to identify the database and specify any optional connection settings. Once you have supplied all the connection parameters by setting the appropriate properties on a ULConnectionParms object, you create a connection string using the ULConnectionParms.ToString. The resulting string is used to create a new ULConnection with the ULConnection(String) constructor or set the ULConnection.ConnectionString of an existing ULConnection object.

Identifying the database

Each instance contains platform-specific paths to the database. Only the value corresponding to the executing platform is used. For example, in the code below the path \UltraLite\mydb1.udb would be used on Windows Mobile, while mydb2.db would be used on other platforms.

```
' Visual Basic
Dim dbName As ULConnectionParms = new ULConnectionParms
dbName.DatabaseOnDevice = "\UltraLite\mydb1.udb"
dbName.DatabaseOnDesktop = "somedir\mydb2.udb"
```

The following code is the C# language equivalent:

```
// C#
ULConnectionParms dbName = new ULConnectionParms();
dbName.DatabaseOnDevice = "\\UltraLite\\mydb1.udb";
dbName.DatabaseOnDesktop = "somedir\mydb2.udb";
```


The recommended extension for UltraLite database files is .udb. On Windows Mobile devices, the default database is \UltraLiteDB\ulstore.udb. On other Windows platforms, the default database is ulstore.udb. In C#, you must escape any backslash characters in paths or use @-quoted string literals.

If you are using multiple databases, you must specify a database name for each database.

Optional connection settings

Depending on your application's needs and how the database was created, you might need to supply a non-default ULConnectionParms.UserID and ULConnectionParms.Password, a database ULConnectionParms.EncryptionKey, and the connection ULConnectionParms.CacheSize. If your application is using multiple connections, you should provide a unique ULConnectionParms.ConnectionName for each connection.

Databases are created with a single authenticated user, DBA, whose initial password is sql. By default, connections are opened using the user ID DBA and password sql. To disable the default user, use the ULConnection.RevokeConnectFrom. To add a user or change a user's password, use the ULConnection.GrantConnectTo.

If an encryption key was supplied when the database was created, all subsequent connections to the database must use the same encryption key. To change a database's encryption key, use the ULConnection.ChangeEncryptionKey.

For more information, see "UltraLite connection parameters" [*UltraLite - Database Management and Reference*].

See also

- "Open method" on page 136
- "DropDatabase method" on page 208
- "ToString method" on page 158
- "ULConnection class" on page 110
- "ConnectionString property" on page 145
- "UserID property" on page 163
- "Password property" on page 162
- "EncryptionKey property" on page 162
- "CacheSize property" on page 160
- "ConnectionName property" on page 160
- "AdditionalParms property" on page 158
- "RevokeConnectFrom method" on page 138
- "GrantConnectTo method" on page 135
- "ChangeEncryptionKey method" on page 121

ULConnectionParms constructor

Initializes a ULConnectionParms instance with its default values.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ULConnectionParms()
```

ToString method

Returns the string representation of this instance.

Visual Basic syntax

```
Public Overrides Function ToString() As String
```

C# syntax

```
public override string ToString()
```

Returns

The string representation of this instance as a semicolon-separated list keyword=value pairs.

AdditionalParms property

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

Visual Basic syntax

```
Public Property AdditionalParms As String
```

C# syntax

```
public string AdditionalParms {get;set;}
```

Exceptions

- **ArgumentException** The value contained an invalid connection string.

Remarks

These parameters are used less frequently.

A semicolon-separated list of keyword=value additional parameters. Values of the keyword=value list must conform to the rules for ULConnection.ConnectionString. The default is a null reference (Nothing in Visual Basic).

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

Additional parameters are:

Keyword	Description
dbn	Identifies a loaded database to which a connection needs to be made. When a database is started, it is assigned a database name, either explicitly with the dbn parameter, or by UltraLite using the base of the file name with the extension and path removed. When opening connections, UltraLite first searches for a running database with a matching dbn. If one is not found, UltraLite starts a new database using the appropriate database file name parameter (DatabaseOnDevice or DatabaseOnDesktop). This parameter is required if the application (or UltraLite engine) needs to access two different databases that have the same base file name. This parameter is only used when opening a connection with ULConnection.Open.
reserve_size	Reserves file system space for storage of UltraLite persistent data. The reserve_size parameter allows you to pre-allocate the file system space required for your UltraLite database without inserting any data. Reserving file system space can improve performance slightly and also prevent out of memory failures. By default, the persistent storage file only grows when required as the application updates the database. Note that reserve_size reserves file system space, which includes the metadata in the persistent store file, and not just the raw data. The metadata overhead and data compression must be considered when deriving the required file system space from the amount of database data. The reserve_size parameter reserves space by growing the persistent store file to the given reserve size on startup, regardless of whether the file previously existed. The file is never truncated. The following parameter string ensures that the persistent store file is at least 2 MB upon startup: createParms.AdditionalParms = "reserve_size=2m" This parameter is only used when opening a connection with ULConnection.Open.
start	Specifies the location and then starts the UltraLite engine. Only supply a StartLine (START) connection parameter if you are connecting to an engine that is not currently running. The location is only required when the UltraLite engine is not in the system path.

For more information, see "UltraLite connection parameters" [*UltraLite - Database Management and Reference*].

See also

- "RuntimeType property" on page 213
- "ConnectionString property" on page 145
- "DatabaseOnDevice property" on page 161
- "DatabaseOnDesktop property" on page 161
- "Open method" on page 136

CacheSize property

Specifies the size of the cache.

Visual Basic syntax

```
Public Property CacheSize As String
```

C# syntax

```
public string CacheSize {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the cache size. The default is a null reference (Nothing in Visual Basic) meaning the default of 16 pages is used.

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

For example, the following sets the cache size to 128 KB.

```
connParms.CacheSize = "128k"
```

The default cache size is 16 pages. Using the default page size of 4 KB, the default cache size is therefore 64 KB. The minimum cache size is platform dependent.

The default cache size is conservative. If your testing shows the need for better performance, you should increase the cache size.

Increasing the cache size beyond the size of the database itself provides no performance improvement and large cache sizes might interfere with the number of other applications you can use.

If the cache size is unspecified or improperly specified, the default size is used.

ConnectionName property

Specifies a name for the connection.

Visual Basic syntax

```
Public Property ConnectionName As String
```

C# syntax

```
public string ConnectionName {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

This is only needed if you create more than one connection to the database.

A string specifying the name of the connection. The default is a null reference (Nothing in Visual Basic).

DatabaseOnDesktop property

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

Visual Basic syntax

```
Public Property DatabaseOnDesktop As String
```

C# syntax

```
public string DatabaseOnDesktop {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the absolute or relative path to the database. If the value is a null reference (Nothing in Visual Basic), the database ulstore.udb is used. In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

DatabaseOnDevice property

Specifies the path and file name of the UltraLite database on Windows Mobile.

Visual Basic syntax

```
Public Property DatabaseOnDevice As String
```

C# syntax

```
public string DatabaseOnDevice {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the full path to the database. If the value is a null reference (Nothing in Visual Basic), the database \UltraLiteDB\ulstore.udb is used. In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

EncryptionKey property

Specifies a key for encrypting the database.

Visual Basic syntax

```
Public Property EncryptionKey As String
```

C# syntax

```
public string EncryptionKey {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the encryption key. The default is a null reference (Nothing in Visual Basic) meaning no encryption.

All connections must use the same key as was specified when the database was created. Lost or forgotten keys result in completely inaccessible databases.

As with all passwords, it is best to choose a key value that cannot be easily guessed. The key can be of arbitrary length, but generally the longer the key, the better, because a shorter key is easier to guess than a longer one. Using a combination of numbers, letters, and special characters decreases the chances of someone guessing the key.

See also

- "ChangeEncryptionKey method" on page 121

Password property

Specifies the password for the authenticated user.

Visual Basic syntax

```
Public Property Password As String
```


C# syntax

```
public string Password {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying a database user ID. The default is a null reference (Nothing in Visual Basic).

Passwords are case sensitive.

When a database is created, the password for the DBA user ID is set to sql.

See also

- "UserID property" on page 163

UserID property

Specifies an authenticated user for the database.

Visual Basic syntax

```
Public Property UserID As String
```

C# syntax

```
public string UserID {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying a database user ID. The default value is a null reference (Nothing in Visual Basic).

User IDs are case-insensitive.

Databases are initially created with a single authenticated user named DBA.

If both the user ID and password are not supplied, the user DBA with password sql are used. To make the database more secure, change the user DBA's password or create new users (using ULConnection.GrantConnectTo) and remove the DBA user (using ULConnection.RevokeConnectFrom).

See also

- "Password property" on page 162
- "GrantConnectTo method" on page 135
- "RevokeConnectFrom method" on page 138

ULConnectionStringBuilder class

Builds a connection string for opening a connection to an UltraLite database.

Visual Basic syntax

```
Public NotInheritable Class ULConnectionStringBuilder
    Inherits System.Data.Common.DbConnectionStringBuilder
```

C# syntax

```
public sealed class ULConnectionStringBuilder :
    System.Data.Common.DbConnectionStringBuilder
```

Base classes

- System.Data.Common.DbConnectionStringBuilder

Members

All members of ULConnectionStringBuilder class, including all inherited members.

Name	Description
"ULConnectionStringBuilder constructor"	Initializes a ULConnectionStringBuilder instance with its default values.
Add method (Inherited from System.Data.Common.DbConnectionStringBuilder)	Adds an entry with the specified key and value into the System.Data.Common.DbConnectionStringBuilder.
AppendKeyValuePair method (Inherited from System.Data.Common.DbConnectionStringBuilder)	Provides an efficient and safe way to append a key and value to an existing System.Text.StringBuilder object.
Clear method (Inherited from System.Data.Common.DbConnectionStringBuilder)	Clears the contents of the System.Data.Common.DbConnectionStringBuilder instance.
ClearPropertyDescriptors method (Inherited from System.Data.Common.DbConnectionStringBuilder)	Clears the collection of System.ComponentModel.PropertyDescriptor objects on the associated System.Data.Common.DbConnectionStringBuilder.
"ContainsKey method"	Determines whether the ULConnectionStringBuilder object contains a specific keyword.

Name	Description
"EquivalentTo method"	Compares the connection information in this ULConnectionStringBuilder object with the connection information in the supplied DbConnectionStringBuilder object.
GetProperties method (Inherited from System.Data.Common.DbConnectionStringBuilder)	Fills a supplied System.Collections.Hashtable with information about all the properties of this System.Data.Common.DbConnectionStringBuilder.
"GetShortName method"	Retrieves the short version of the supplied keyword.
"Remove method"	Removes the entry with the specified key from the ULConnectionStringBuilder instance.
ShouldSerialize method (Inherited from System.Data.Common.DbConnectionStringBuilder)	Indicates whether the specified key exists in this System.Data.Common.DbConnectionStringBuilder instance.
ToString method (Inherited from System.Data.Common.DbConnectionStringBuilder)	Returns the connection string associated with this System.Data.Common.DbConnectionStringBuilder.
"TryGetValue method"	Retrieves a value corresponding to the supplied key from this ULConnectionStringBuilder.
BrowsableConnectionString property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets or sets a value that indicates whether the System.Data.Common.DbConnectionStringBuilder.ConnectionString property is visible in Visual Studio designers.
"CacheSize property"	**UL Ext:** Specifies the size of the cache.
"ConnectionName property"	Specifies a name for the connection.
ConnectionString property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets or sets the connection string associated with the System.Data.Common.DbConnectionStringBuilder.
Count property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets the current number of keys that are contained within the System.Data.Common.DbConnectionStringBuilder.ConnectionString property.
"DatabaseKey property"	Specifies a key for encrypting the database.
"DatabaseName property"	Specifies a name for the database or the name of a loaded database to which a connection needs to be made.

Name	Description
"DatabaseOnDesktop property"	**UL Ext:** Specifies the path and file name of the UltraLite database on Windows desktop platforms.
"DatabaseOnDevice property"	**UL Ext:** Specifies the path and file name of the UltraLite database on Windows Mobile.
IsFixedSize property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets a value that indicates whether the System.Data.Common.DbConnectionStringBuilder has a fixed size.
IsReadOnly property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets a value that indicates whether the System.Data.Common.DbConnectionStringBuilder is read-only.
Keys property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets an System.Collections.ICollection that contains the keys in the System.Data.Common.DbConnectionStringBuilder.
"OrderedTableScans property"	Specifies whether SQL queries without ORDER BY clauses should perform ordered table scans by default.
"Password property"	Specifies the password for the authenticated user.
"ReserveSize property"	**UL Ext:** Specifies the reserve file system space for storage of UltraLite persistent data.
"StartLine property"	Specifies the location and then starts the UltraLite engine.
"this property"	Specifies the value of the specified connection keyword.
"UserID property"	Specifies an authenticated user for the database.
Values property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets an System.Collections.ICollection that contains the values in the System.Data.Common.DbConnectionStringBuilder.

Remarks

The frequently-used connection parameters are individual properties on the ULConnectionStringBuilder object.

The ULConnectionStringBuilder class is not available in the .NET Compact Framework 2.0.

A ULConnectionStringBuilder object is used to specify the parameters for opening a connection (ULConnection.Open) or dropping a database (ULDatabaseManager.DropDatabase).

Leading and trailing spaces are ignored in all values. Values must not contain leading or trailing spaces, or a semicolon, or begin with either a single quote or a double quote.

When building a connection string, you need to identify the database and specify any optional connection settings. Once you have supplied all the connection parameters by setting the appropriate properties on a ULConnectionStringBuilder object, you create a connection string using the System.Data.Common.DbConnectionStringBuilder.ConnectionString. The resulting string is used to create a new ULConnection with the ULConnection(String) constructor or set the ULConnection.ConnectionString of an existing ULConnection object.

Identifying the database

Each instance contains platform-specific paths to the database. Only the value corresponding to the executing platform is used. For example, in the code below the path \UltraLite\mydb1.udb would be used on Windows Mobile, while mydb2.db would be used on other platforms.

```
' Visual Basic
Dim dbName As ULConnectionStringBuilder = _
    new ULConnectionStringBuilder
dbName.DatabaseOnDevice = "\UltraLite\mydb1.udb"
dbName.DatabaseOnDesktop = "somedir\mydb2.udb"
```

The following code is the C# language equivalent:

```
// C#
ULConnectionStringBuilder dbName = new ULConnectionStringBuilder();
dbName.DatabaseOnDevice = "\\UltraLite\\mydb1.udb";
 dbName.DatabaseOnDesktop = @"somedir\\mydb2.udb";
```

The recommended extension for UltraLite database files is .udb. On Windows Mobile devices, the default database is \UltraLiteDB\ulstore.udb. On other Windows platforms, the default database is ulstore.udb. In C#, you must escape any backslash characters in paths or use @-quoted string literals.

If you are using multiple databases, you must specify a database name for each database.

Optional connection settings

Depending on your application's needs and how the database was created, you might need to supply a non-default ULConnectionStringBuilder.UserID and ULConnectionStringBuilder.Password, a database ULConnectionStringBuilder.DatabaseKey, and the connection ULConnectionStringBuilder.CacheSize. If your application is using multiple connections, you should provide a unique ULConnectionStringBuilder.ConnectionName for each connection.

Databases are created with a single authenticated user, DBA, whose initial password is sql. By default, connections are opened using the user ID DBA and password sql. To disable the default user, use the ULConnection.RevokeConnectFrom. To add a user or change a user's password, use the ULConnection.GrantConnectTo.

If an encryption key was supplied when the database was created, all subsequent connections to the database must use the same encryption key. To change a database's encryption key, use the ULConnection.ChangeEncryptionKey.

For more information, see "UltraLite connection parameters" [*UltraLite - Database Management and Reference*].

See also

- "Open method" on page 136
- "DropDatabase method" on page 208
- "ULConnection class" on page 110
- "ConnectionString property" on page 145
- "DatabaseName property" on page 173
- "UserID property" on page 178
- "Password property" on page 175
- "DatabaseKey property" on page 173
- "CacheSize property" on page 171
- "ConnectionName property" on page 172
- "RevokeConnectFrom method" on page 138
- "GrantConnectTo method" on page 135
- "ChangeEncryptionKey method" on page 121
- System.Data.Common.DbConnectionStringBuilder.ConnectionString

ULConnectionStringBuilder constructor

Initializes a ULConnectionStringBuilder instance with its default values.

Overload list

Name	Description
"ULConnectionStringBuilder() constructor"	Initializes a ULConnectionStringBuilder instance with its default values.
"ULConnectionStringBuilder(string) constructor"	Initializes a ULConnectionStringBuilder instance with the specified connection string.

ULConnectionStringBuilder() constructor

Initializes a ULConnectionStringBuilder instance with its default values.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ULConnectionStringBuilder()
```


ULConnectionStringBuilder(string) constructor

Initializes a ULConnectionStringBuilder instance with the specified connection string.

Visual Basic syntax

```
Public Sub New(ByVal connectionString As String)
```

C# syntax

```
public ULConnectionStringBuilder(string connectionString)
```

Parameters

- **connectionString** An UltraLite.NET connection string. A connection string is a semicolon-separated list of keyword-value pairs.

ContainsKey method

Determines whether the ULConnectionStringBuilder object contains a specific keyword.

Visual Basic syntax

```
Public Overrides Function ContainsKey(
    ByVal keyword As String
) As Boolean
```

C# syntax

```
public override bool ContainsKey(string keyword)
```

Parameters

- **keyword** The name of the connection keyword.

Returns

True if this connection string builder contains a value for the specified keyword, otherwise returns false.

EquivalentTo method

Compares the connection information in this ULConnectionStringBuilder object with the connection information in the supplied DbConnectionStringBuilder object.

Visual Basic syntax

```
Public Overrides Function EquivalentTo(
    ByVal connectionStringBuilder As DbConnectionStringBuilder
) As Boolean
```

C# syntax

```
public override bool EquivalentTo(
    DbConnectionStringBuilder connectionStringBuilder
)
```

Parameters

- **connectionStringBuilder** The other DbConnectionStringBuilder object to compare this ULConnectionStringBuilder object to.

Returns

True if this object is equivalent to the specified DbConnectionStringBuilder object, otherwise returns false.

See also

- System.Data.Common.DbConnectionStringBuilder

GetShortName method

Retrieves the short version of the supplied keyword.

Visual Basic syntax

```
Public Shared Function GetShortName(ByVal keyword As String) As String
```

C# syntax

```
public static string GetShortName(string keyword)
```

Parameters

- **keyword** The key of the item to retrieve.

Returns

The short version of the supplied keyword if keyword is recognized, null otherwise.

Remove method

Removes the entry with the specified key from the ULConnectionStringBuilder instance.

Visual Basic syntax

```
Public Overrides Function Remove(ByVal keyword As String) As Boolean
```

C# syntax

```
public override bool Remove(string keyword)
```

Parameters

- **keyword** The name of the connection keyword.

Returns

True if the key existed within the connection string and was removed; false if the key did not exist.

TryGetValue method

Retrieves a value corresponding to the supplied key from this ULConnectionStringBuilder.

Visual Basic syntax

```
Public Overrides Function TryGetValue(
    ByVal keyword As String,
    ByVal value As Object
) As Boolean
```

C# syntax

```
public override bool TryGetValue(string keyword, out Object value)
```

Parameters

- **keyword** The key of the item to retrieve.
- **value** The value corresponding to the key.

Returns

True if keyword was found within the connection string, false otherwise.

Remarks

The TryGetValue method lets developers safely retrieve a value from a ULConnectionStringBuilder without needing to first call the ContainsKey method. Because TryGetValue does not raise an exception when you call it, passing in a nonexistent key, you do not have to look for a key before retrieving its value. Calling TryGetValue with a nonexistent key places the null value (Nothing in Visual Basic) in the value parameter.

CacheSize property

UL Ext: Specifies the size of the cache.

Visual Basic syntax

```
Public Property CacheSize As String
```

C# syntax

```
public string CacheSize {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the cache size. The default is a null reference (Nothing in Visual Basic) meaning the default of 16 pages is used.

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

For example, the following sets the cache size to 128 KB.

```
connParms.CacheSize = "128k"
```

The default cache size is 16 pages. Using the default page size of 4 KB, the default cache size is therefore 64 KB. The minimum cache size is platform dependent.

The default cache size is conservative. If your testing shows the need for better performance, you should increase the cache size.

Increasing the cache size beyond the size of the database itself provides no performance improvement and large cache sizes might interfere with the number of other applications you can use.

If the cache size is unspecified or improperly specified, the default size is used.

ConnectionName property

Specifies a name for the connection.

Visual Basic syntax

```
Public Property ConnectionName As String
```

C# syntax

```
public string ConnectionName {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

This is only needed if you create more than one connection to the database.

A string specifying the name of the connection. The default is a null reference (Nothing in Visual Basic).

DatabaseKey property

Specifies a key for encrypting the database.

Visual Basic syntax

```
Public Property DatabaseKey As String
```

C# syntax

```
public string DatabaseKey {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the encryption key. The default is a null reference (Nothing in Visual Basic) meaning no encryption.

All connections must use the same key as was specified when the database was created. Lost or forgotten keys result in completely inaccessible databases.

As with all passwords, it is best to choose a key value that cannot be easily guessed. The key can be of arbitrary length, but generally the longer the key, the better, because a shorter key is easier to guess than a longer one. Using a combination of numbers, letters, and special characters decreases the chances of someone guessing the key.

See also

- "ChangeEncryptionKey method" on page 121

DatabaseName property

Specifies a name for the database or the name of a loaded database to which a connection needs to be made.

Visual Basic syntax

```
Public Property DatabaseName As String
```

C# syntax

```
public string DatabaseName {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the name of the database. The default is a null reference (Nothing in Visual Basic).

When a database is started, it is assigned a database name, either explicitly with the dbn parameter, or by UltraLite using the base of the file name with the extension and path removed.

When opening connections, UltraLite first searches for a running database with a matching dbn. If one is not found, UltraLite starts a new database using the appropriate database file name parameter (DatabaseOnDevice or DatabaseOnDesktop).

This parameter is required if the application (or UltraLite engine) needs to access two different databases that have the same base file name.

See also

- "DatabaseOnDevice property" on page 174
- "DatabaseOnDesktop property" on page 174

DatabaseOnDesktop property

UL Ext: Specifies the path and file name of the UltraLite database on Windows desktop platforms.

Visual Basic syntax

```
Public Property DatabaseOnDesktop As String
```

C# syntax

```
public string DatabaseOnDesktop {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the absolute or relative path to the database. If the value is a null reference (Nothing in Visual Basic), the database ulstore.udb is used. In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

DatabaseOnDevice property

UL Ext: Specifies the path and file name of the UltraLite database on Windows Mobile.

Visual Basic syntax

```
Public Property DatabaseOnDevice As String
```

C# syntax

```
public string DatabaseOnDevice {get;set;}
```


Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the full path to the database. If the value is a null reference (Nothing in Visual Basic), the database \UltraLiteDB\ulstore.udb is used. In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

OrderedTableScans property

Specifies whether SQL queries without ORDER BY clauses should perform ordered table scans by default.

Visual Basic syntax

```
Public Property OrderedTableScans As String
```

C# syntax

```
public string OrderedTableScans {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A boolean string specifying whether to use ordered table scans or not. For example, true/false, yes/no, 1/0, and so on. The default value is a null reference (Nothing in Visual Basic).

When using dynamic SQL in UltraLite, if order is not important for executing a query, UltraLite accesses the rows directly from the database pages rather than using the primary key index. This improves performance of fetching rows. To use this optimization, the query must be read only and must scan all the rows.

When rows are expected in a specific order, an ORDER BY statement should be included as part of the SQL query. However, it's possible that some applications have come to rely on the behavior that defaults to returning rows in the primary key order. In this case, users should set the OrderedTableScans parameter to 1 (true, yes, on) to revert to the old behavior when iterating over a table.

When OrderedTableScans is set to 1 (true, yes, on) and the user does not specify an ORDER BY clause or if a query would not benefit from an index, UltraLite defaults to using the primary key.

Password property

Specifies the password for the authenticated user.

Visual Basic syntax

```
Public Property Password As String
```

C# syntax

```
public string Password {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying a database user ID. The default is a null reference (Nothing in Visual Basic).

Passwords are case sensitive.

When a database is created, the password for the DBA user ID is set to sql.

See also

- "UserID property" on page 178

ReserveSize property

UL Ext: Specifies the reserve file system space for storage of UltraLite persistent data.

Visual Basic syntax

```
Public Property ReserveSize As String
```

C# syntax

```
public string ReserveSize {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the reserve size. The default is a null reference (Nothing in Visual Basic).

The values for the reserve size parameter is specified in units of bytes. Use the suffix k or K to indicate units of kilobytes and the suffix m or M to indicate megabytes.

The reserve_size parameter allows you to pre-allocate the file system space required for your UltraLite database without inserting any data. Reserving file system space can improve performance slightly and also prevent out of memory failures. By default, the persistent storage file only grows when required as the application updates the database.

Note that reserve_size reserves file system space, which includes the metadata in the persistent store file, and not just the raw data. The metadata overhead and data compression must be considered when deriving the required file system space from the amount of database data.

The reserve_size parameter reserves space by growing the persistent store file to the given reserve size on startup, regardless of whether the file previously existed. The file is never truncated.

The following parameter string ensures that the persistent store file is at least 2 MB upon startup.

```
connParms.ReserveSize = "2m"
```

StartLine property

Specifies the location and then starts the UltraLite engine.

Visual Basic syntax

```
Public Property StartLine As String
```

C# syntax

```
public string StartLine {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the location of the UltraLite engine executable. The default value is a null reference (Nothing in Visual Basic).

Only supply a StartLine (START) connection parameter if you are connecting to an engine that is not currently running.

See also

- "RuntimeType property" on page 213

this property

Specifies the value of the specified connection keyword.

Visual Basic syntax

```
Public Overrides Property Item(ByVal keyword As String) As Object
```

C# syntax

```
public override object this[string keyword] {get;set;}
```

Parameters

- **keyword** The name of the connection keyword.

Remarks

An object representing the value of the specified connection keyword.

Connection keywords and the corresponding properties on ULConnectionStringBuilder are described in the table below:

Keyword	Corresponding Property
cache_size	ULConnectionStringBuilder.CacheSize
ce_file	ULConnectionStringBuilder.DatabaseOnDevice
con	ULConnectionStringBuilder.ConnectionName
dbkey	ULConnectionStringBuilder.DatabaseKey
dbn	ULConnectionStringBuilder.DatabaseName
nt_file	ULConnectionStringBuilder.DatabaseOnDesktop
pwd	ULConnectionStringBuilder.Password
reserve_size	ULConnectionStringBuilder.ReserveSize
start	ULConnectionStringBuilder.StartLine
uid	ULConnectionStringBuilder.UserID

See also

- “CacheSize property” on page 171
- “DatabaseOnDevice property” on page 174
- “ConnectionName property” on page 172
- “DatabaseKey property” on page 173
- “DatabaseName property” on page 173
- “DatabaseOnDesktop property” on page 174
- “Password property” on page 175
- “ReserveSize property” on page 176
- “StartLine property” on page 177
- “UserID property” on page 178

UserID property

Specifies an authenticated user for the database.

Visual Basic syntax

```
Public Property UserID As String
```

C# syntax

```
public string UserID {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying a database user ID. The default value is a null reference (Nothing in Visual Basic).

User IDs are case-insensitive.

Databases are initially created with a single authenticated user named DBA.

If both the user ID and password are not supplied, the user DBA with password sql are used. To make the database more secure, change the user DBA's password or create new users (using ULConnection.GrantConnectTo) and remove the DBA user (using ULConnection.RevokeConnectFrom).

See also

- "Password property" on page 175
- "GrantConnectTo method" on page 135
- "RevokeConnectFrom method" on page 138

ULCreateParms class

UL Ext: Builds a string of creation-time options for creating an UltraLite database.

Visual Basic syntax

```
Public Class ULCreateParms
```

C# syntax

```
public class ULCreateParms
```

Members

All members of ULCreateParms class, including all inherited members.

Name	Description
"ULCreateParms constructor"	Initializes a ULCreateParms instance with its default values.
"ToString method"	Returns the string representation of this instance.

Name	Description
“CaseSensitive property”	Specifies whether the new database should be case sensitive when comparing string values.
“ChecksumLevel property”	Specifies the level of database page checksums enabled for the new database.
“DateFormat property”	Specifies the date format used for string conversions by the new database.
“DateOrder property”	Specifies the date order used for string conversions by the new database.
“FIPS property”	Specifies whether the new database should be using AES_FIPS encryption or AES encryption.
“MaxHashSize property”	Specifies the default maximum number of bytes to use for index hashing in the new database.
“NearestCentury property”	Specifies the nearest century used for string conversions by the new database.
“Obfuscate property”	Specifies whether the new database should be using obfuscation (simple encryption) or not.
“PageSize property”	Specifies the page size of the new database, in bytes or kilobytes.
“Precision property”	Specifies the floating-point precision used for string conversions by the new database.
“Scale property”	Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION during string conversions by the new database.
“TimeFormat property”	Specifies the time format used for string conversions by the new database.
“TimestampFormat property”	Specifies the timestamp format used for string conversions by the new database.
“TimestampIncrement property”	Specifies the minimum difference between two unique timestamps, in microseconds (1,000,000th of a second).
“UTF8Encoding property”	Specifies whether the new database should be using the UTF8 character set or the character set associated with the collation.

Remarks

A ULCreateParms object is used to specify the parameters for creating a database (ULDatabaseManager.CreateDatabase(string,string)).

Leading and trailing spaces are ignored in all string values. Values must not contain leading or trailing spaces, or a semicolon, or begin with either a single quote or a double quote.

Once you have supplied all the creation parameters by setting the appropriate properties on a ULCreateParms object, you create a creation parameters string using the ULCreateParms.ToString. The resulting string can then be used as the createParms parameter of the ULDatabaseManager.CreateDatabase(string,string) method.

For more information, see "UltraLite connection parameters" [*UltraLite - Database Management and Reference*].

See also

- "GetDatabaseProperty method" on page 214
- "CreateDatabase method" on page 207
- "ToString method" on page 182

Example

The following code creates the database \UltraLite\MyDatabase.udb on a Windows Mobile device. The database is created case sensitive and with the UTF8 character set.

```
' Visual Basic
Dim createParms As ULCreateParms = New ULCreateParms
createParms.CaseSensitive = True
createParms.UTF8Encoding = True
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"

ULConnection.DatabaseManager.CreateDatabase( _
    openParms.ToString(), _
    createParms.ToString() _
)

Dim conn As ULConnection = _
    New ULConnection( openParms.ToString() )
conn.Open()
```

The following code is the C# language equivalent:

```
// C#
ULCreateParms createParms = new ULCreateParms();
createParms.CaseSensitive = true;
createParms.UTF8Encoding = true;
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnDevice = ".udb";

ULConnection.DatabaseManager.CreateDatabase(
    openParms.ToString(),
    createParms.ToString()
);

ULConnection conn = new ULConnection( openParms.ToString() );
conn.Open();
```

ULCreateParms constructor

Initializes a ULCreateParms instance with its default values.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ULCreateParms()
```

ToString method

Returns the string representation of this instance.

Visual Basic syntax

```
Public Overrides Function ToString() As String
```

C# syntax

```
public override string ToString()
```

Returns

The string representation of this instance as a semicolon-separated list keyword=value pairs.

CaseSensitive property

Specifies whether the new database should be case sensitive when comparing string values.

Visual Basic syntax

```
Public Property CaseSensitive As Boolean
```

C# syntax

```
public bool CaseSensitive {get;set;}
```

Remarks

True if the database should be case sensitive, false if the database should be case insensitive. The default is false.

CaseSensitive only affects how string data is compared and sorted. Database identifiers such as table names, column names, index names, and connection user IDs are always case insensitive. Connection passwords and database encryption keys are always case sensitive.

ChecksumLevel property

Specifies the level of database page checksums enabled for the new database.

Visual Basic syntax

```
Public Property ChecksumLevel As Integer
```

C# syntax

```
public int ChecksumLevel {get;set;}
```

Exceptions

- **ArgumentException** The value is invalid.

Remarks

An integer specifying the checksum level. Valid values are 0, 1, and 2. The default is 0.

DateFormat property

Specifies the date format used for string conversions by the new database.

Visual Basic syntax

```
Public Property DateFormat As String
```

C# syntax

```
public string DateFormat {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the date format. If the value is a null reference (Nothing in Visual Basic), the database uses "YYYY-MM-DD". In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

DateOrder property

Specifies the date order used for string conversions by the new database.

Visual Basic syntax

```
Public Property DateOrder As ULDateOrder
```

C# syntax

```
public ULDateOrder DateOrder {get;set;}
```

Remarks

A ULDateOrder value identifying the date order for string conversions. The default is YMD.

See also

- "ULDateOrder enumeration" on page 437

FIPS property

Specifies whether the new database should be using AES_FIPS encryption or AES encryption.

Visual Basic syntax

```
Public Property FIPS As Boolean
```

C# syntax

```
public bool FIPS {get;set;}
```

Remarks

True if the database should be encrypted using AES_FIPS, false if the database should be encrypted with AES. The default is false.

Encryption must be turned on by supplying a value for the connection parameter EncryptionKey when the new database is created. If FIPS is set true and no encryption key is supplied, the ULDatabaseManager.CreateDatabase(string,string) method fails with a missing encryption key error.

See also

- "EncryptionKey property" on page 162
- "CreateDatabase method" on page 207

MaxHashSize property

Specifies the default maximum number of bytes to use for index hashing in the new database.

Visual Basic syntax

```
Public Property MaxHashSize As Integer
```

C# syntax

```
public int MaxHashSize {get;set;}
```

Exceptions

- **ArgumentException** The value is invalid.

Remarks

An integer specifying the maximum hash size. The value must be in the range [0,32]. The default is 8.

NearestCentury property

Specifies the nearest century used for string conversions by the new database.

Visual Basic syntax

```
Public Property NearestCentury As Integer
```

C# syntax

```
public int NearestCentury {get;set;}
```

Exceptions

- **ArgumentException** The value is invalid.

Remarks

An integer specifying the nearest century. The value must be in the range [0,100]. The default is 50.

Obfuscate property

Specifies whether the new database should be using obfuscation (simple encryption) or not.

Visual Basic syntax

```
Public Property Obfuscate As Boolean
```

C# syntax

```
public bool Obfuscate {get;set;}
```

Remarks

True if the database should be encrypted using obfuscation, false if the database should not be obfuscated. The default is false.

This option is ignored if FIPS encryption is turned on (ULCreateParms.FIPS). If obfuscation is turned on and a value is supplied for the connection parameter EncryptionKey (DBKEY) when the new database is created, the encryption key is ignored.

See also

- "FIPS property" on page 184

PageSize property

Specifies the page size of the new database, in bytes or kilobytes.

Visual Basic syntax

```
Public Property PageSize As Integer
```

C# syntax

```
public int PageSize {get;set;}
```

Exceptions

- **ArgumentException** The value is invalid.

Remarks

An integer specifying the page size in bytes. Valid values are 1024 (1K), 2048 (2K), 4096 (4K), 8192 (8K), 16384 (16K). The default is 4096.

Precision property

Specifies the floating-point precision used for string conversions by the new database.

Visual Basic syntax

```
Public Property Precision As Integer
```

C# syntax

```
public int Precision {get;set;}
```

Exceptions

- **ArgumentException** The value is invalid.

Remarks

An integer specifying the precision. The value must be in the range [1,127]. The default is 30.

See also

- "Scale property" on page 186

Scale property

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION during string conversions by the new database.

Visual Basic syntax

```
Public Property Scale As Integer
```


C# syntax

```
public int Scale {get;set;}
```

Exceptions

- **ArgumentException** The value is invalid.

Remarks

An integer specifying the scale. The value must be in the range [0,127]. The default is 6.

Scale must be less than or equal to the Precision. If Scale is greater than the Precision, an error occurs when creating the database.

TimeFormat property

Specifies the time format used for string conversions by the new database.

Visual Basic syntax

```
Public Property TimeFormat As String
```

C# syntax

```
public string TimeFormat {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the time format. If the value is a null reference (Nothing in Visual Basic), the database uses "HH:NN:SS.SSS". In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

TimestampFormat property

Specifies the timestamp format used for string conversions by the new database.

Visual Basic syntax

```
Public Property TimestampFormat As String
```

C# syntax

```
public string TimestampFormat {get;set;}
```

Exceptions

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the timestamp format. If the value is a null reference (Nothing in Visual Basic), the database uses "YYYY-MM-DD HH:NN:SS.SSS". In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

TimestampIncrement property

Specifies the minimum difference between two unique timestamps, in microseconds (1,000,000th of a second).

Visual Basic syntax

```
Public Property TimestampIncrement As Integer
```

C# syntax

```
public int TimestampIncrement {get;set;}
```

Exceptions

- **ArgumentException** The value is invalid.

Remarks

An integer specifying the timestamp increment. The value must be in the range [1,60000000]. The default is 1.

UTF8Encoding property

Specifies whether the new database should be using the UTF8 character set or the character set associated with the collation.

Visual Basic syntax

```
Public Property UTF8Encoding As Boolean
```

C# syntax

```
public bool UTF8Encoding {get;set;}
```

Remarks

True if the database should use the UTF8 character set, false if the database should use the character set associated with the collation. The default is false.

Choose to use the UTF8 character set if you want to store characters that are not in the character set associated with the collation. For example, you create a database with the 1252LATIN1 collation because you want US sorting but specify UTF8Encoding true because you want to store international addresses as they are spelled locally.

For databases used on Symbian OS devices, you must set UTF8Encoding true.

ULCursorSchema class

UL Ext: Represents the schema of an UltraLite.NET cursor.

Visual Basic syntax

```
Public MustInherit Class ULCursorSchema
```

C# syntax

```
public abstract class ULCursorSchema
```

Derived classes

- "ULResultSetSchema class" on page 358
- "ULTableSchema class" on page 417

Members

All members of ULCursorSchema class, including all inherited members.

Name	Description
"GetColumnID method"	Returns the column ID of the named column.
"GetColumnName method"	Returns the name of the column identified by the specified column ID.
"GetColumnPrecision method"	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnScale method"	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnSize method"	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
"GetColumnSQLName method"	Returns the name of the column identified by the specified column ID.
"GetColumnULDbType method"	Returns the UltraLite.NET data type of the column identified by the specified column ID.

Name	Description
"GetSchemaTable method"	Returns a System.Data.DataTable that describes the column schema of the ULDataReader.
"ColumnCount property"	Returns the number of columns in the cursor.
"IsOpen property"	Checks whether the cursor schema is currently open.
"Name property"	Returns the name of the cursor.

Remarks

This class is an abstract base class of the ULTableSchema class and the ULResultSetSchema class.

Note
For users porting from the iAnywhere.UltraLite namespace, Column IDs are 0-based, not 1-based as they are in the iAnywhere.UltraLite namespace.

See also

- "ULTableSchema class" on page 417
- "ULResultSetSchema class" on page 358

GetColumnID method

Returns the column ID of the named column.

Visual Basic syntax

```
Public Function GetColumnID(ByVal name As String) As Short
```

C# syntax

```
public short GetColumnID(string name)
```

Parameters

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

Returns

The column ID of the named column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Column IDs range from 0 to ColumnCount-1, inclusive.

Note that in result sets, not all columns have names and not all column names are unique. If you are not using aliases, the name of a non-computed column is prefixed with the name of the table the column is from. For example, MyTable.ID is the name of the only column in the result set for the query "SELECT ID FROM MyTable".

Column IDs and counts might change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

See also

- "ColumnCount property" on page 195

GetColumnName method

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

Visual Basic syntax

```
Public Function GetColumnName(ByVal columnID As Integer) As String
```

C# syntax

```
public string GetColumnName(int columnID)
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Returns

The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Note that in result sets, not all columns have names and not all column names are unique. If you are not using aliases, the name of a non-computed column is prefixed with the name of the table the column is from. For example, MyTable.ID is the name of the only column in the result set for the query "SELECT ID FROM MyTable".

Column IDs and count may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

See also

- "ColumnCount property" on page 195

GetColumnPrecision method

Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).

Visual Basic syntax

```
Public Function GetColumnPrecision(ByVal columnID As Integer) As Integer
```

C# syntax

```
public int GetColumnPrecision(int columnID)
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Returns

The precision of the specified numeric column.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetColumnULDbType method" on page 194
- "ColumnCount property" on page 195

GetColumnScale method

Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).

Visual Basic syntax

```
Public Function GetColumnScale(ByVal columnID As Integer) As Integer
```

C# syntax

```
public int GetColumnScale(int columnID)
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Returns

The scale of the specified numeric column.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetColumnULDbType method" on page 194
- "ColumnCount property" on page 195

GetColumnSize method

Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).

Visual Basic syntax

```
Public Function GetColumnSize(ByVal columnID As Integer) As Integer
```

C# syntax

```
public int GetColumnSize(int columnID)
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Returns

The size of the specified sized column.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetColumnULDbType method" on page 194
- "ColumnCount property" on page 195

GetColumnSQLName method

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

Visual Basic syntax

```
Public Function GetColumnSQLName(ByVal columnID As Integer) As String
```

C# syntax

```
public string GetColumnSQLName(int columnID)
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Returns

The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Note that in result sets, not all columns have names and not all column names are unique. If you are using aliases, the name of the column is the alias.

The GetColumnSQLName method differs from the GetColumnName in that for non-aliased, non-computed columns GetColumnSQLName always returns just the name of the column (without the table name as a prefix). While this behavior more closely resembles the behavior of other ADO.NET providers, it is more likely to produce non-unique names.

Column IDs and count may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

See also

- "ColumnCount property" on page 195
- "GetColumnName method" on page 191
- "ColumnCount property" on page 195

GetColumnULDbType method

Returns the UltraLite.NET data type of the column identified by the specified column ID.

Visual Basic syntax

```
Public Function GetColumnULDbType(ByVal columnID As Integer) As ULDbType
```

C# syntax

```
public ULDbType GetColumnULDbType(int columnID)
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Returns

A ULDbType enumerated integer.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ColumnCount property" on page 195
- "ULDbType enumeration" on page 437

GetSchemaTable method

Returns a System.Data.DataTable that describes the column schema of the ULDataReader.

Visual Basic syntax

```
Public Function GetSchemaTable() As DataTable
```

C# syntax

```
public DataTable GetSchemaTable()
```

Returns

A System.Data.DataTable that describes the column schema.

See also

- "GetSchemaTable method" on page 241
- "ULDataReader class" on page 220
- System.Data.DataTable

ColumnCount property

Returns the number of columns in the cursor.

Visual Basic syntax

```
Public ReadOnly Property ColumnCount As Short
```

C# syntax

```
public short ColumnCount {get;}
```

Remarks

The number of columns in the cursor or 0 if the cursor schema is closed.

Column IDs range from 0 to ColumnCount-1, inclusive.

Column IDs and count might change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

IsOpen property

Checks whether the cursor schema is currently open.

Visual Basic syntax

```
Public ReadOnly Property IsOpen As Boolean
```

C# syntax

```
public bool IsOpen {get;}
```

Remarks

True if the cursor schema is currently open, false if the cursor schema is closed.

Name property

Returns the name of the cursor.

Visual Basic syntax

```
Public ReadOnly Property Name As String
```

C# syntax

```
public abstract string Name {get;}
```

Remarks

The name of the cursor as a string.

ULDataAdapter class

Represents a set of commands and a database connection used to fill a System.Data.DataSet and to update a database.

Visual Basic syntax

```
Public NotInheritable Class ULDataAdapter
    Inherits System.Data.Common.DbDataAdapter
    Implements System.ICloneable
```

C# syntax

```
public sealed class ULDataAdapter :
    System.Data.Common.DbDataAdapter,
    System.ICloneable
```

Base classes

- System.Data.Common.DbDataAdapter
- System.ICloneable

Members

All members of ULDataAdapter class, including all inherited members.

Name	Description
"ULDataAdapter constructor"	Initializes a ULDataAdapter object.
AddToBatch method (Inherited from System.Data.Common.DbDataAdapter)	Adds a System.Data.IDbCommand to the current batch.
ClearBatch method (Inherited from System.Data.Common.DbDataAdapter)	Removes all System.Data.IDbCommand objects from the batch.
CreateRowUpdatedEvent method (Inherited from System.Data.Common.DbDataAdapter)	Initializes a new instance of the System.Data.Common.RowUpdatedEventArgs class.
CreateRowUpdatingEvent method (Inherited from System.Data.Common.DbDataAdapter)	Initializes a new instance of the System.Data.Common.RowUpdatingEventArgs class.
Dispose method (Inherited from System.Data.Common.DbDataAdapter)	Releases the unmanaged resources used by the System.Data.Common.DbDataAdapter and optionally releases the managed resources.
ExecuteBatch method (Inherited from System.Data.Common.DbDataAdapter)	Executes the current batch.
Fill method (Inherited from System.Data.Common.DbDataAdapter)	Adds or refreshes rows in the System.Data.DataSet.
FillSchema method (Inherited from System.Data.Common.DbDataAdapter)	Adds a System.Data.DataTable named "Table" to the specified System.Data.DataSet and configures the schema to match that in the data source based on the specified System.Data.SchemaType.
GetBatchedParameter method (Inherited from System.Data.Common.DbDataAdapter)	Returns a System.Data.IDataParameter from one of the commands in the current batch.
GetBatchedRecordsAffected method (Inherited from System.Data.Common.DbDataAdapter)	Returns information about an individual update attempt within a larger batched update.
"GetFillParameters method"	Returns the parameters set by the user when executing a SELECT statement.

Name	Description
InitializeBatching method (Inherited from System.Data.Common.DbDataAdapter)	Initializes batching for the System.Data.Common.DbDataAdapter.
FillError (Inherited from System.Data.Common.DbDataAdapter)OnFillError method (Inherited from System.Data.Common.DbDataAdapter)	
OnRowUpdated method (Inherited from System.Data.Common.DbDataAdapter)	Raises the RowUpdated event of a .NET Framework data provider.
OnRowUpdating method (Inherited from System.Data.Common.DbDataAdapter)	Raises the RowUpdating event of a .NET Framework data provider.
TerminateBatching method (Inherited from System.Data.Common.DbDataAdapter)	Ends batching for the System.Data.Common.DbDataAdapter.
Update method (Inherited from System.Data.Common.DbDataAdapter)	Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified array of System.Data.DataRow objects.
“DeleteCommand property”	Specifies a ULCommand object that is executed against the database when DbDataAdapter.Update(System.Data.DataSet) is called to delete rows in the database that correspond to deleted rows in the System.Data.DataSet.
FillCommandBehavior property (Inherited from System.Data.Common.DbDataAdapter)	Gets or sets the behavior of the command used to fill the data adapter.
“InsertCommand property”	Specifies a ULCommand object that is executed against the database when DbDataAdapter.Update(System.Data.DataSet) is called to insert rows in the database that correspond to inserted rows in the System.Data.DataSet.
“SelectCommand property”	Specifies a ULCommand that is used during System.Data.Common.DbDataAdapter.Fill(System.Data.DataSet) or System.Data.Common.DbDataAdapter.FillSchema(System.Data.DataSet,System.Data.SchemaType) to obtain a result set from the database for copying into a System.Data.DataSet.

Name	Description
"TableMappings property"	Returns a collection that provides the master mapping between a source table and a System.Data.DataTable
UpdateBatchSize property (Inherited from System.Data.Common.DbDataAdapter)	Gets or sets a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.
"UpdateCommand property"	Specifies a ULCommand object that is executed against the database when System.Data.Common.DbDataAdapter.Update(System.Data.DataSet) is called to update rows in the database that correspond to updated rows in the System.Data.DataSet.
"RowUpdated event"	Occurs during an update after a command is executed against the data source.
"RowUpdating event"	Occurs during an update before a command is executed against the data source.
DefaultSourceTableName field (Inherited from System.Data.Common.DbDataAdapter)	The default name used by the System.Data.Common.DataAdapter object for table mappings.

Remarks

The System.Data.DataSet provides a way to work with data offline; that is, away from your UltraLite database. The ULDataAdapter provides methods to associate a System.Data.DataSet with a set of SQL statements.

Since UltraLite is a local database and MobiLink has conflict resolution, the use of the ULDataAdapter is limited. For most purposes, the ULDataReader or the ULTable provide more efficient access to data.

See also

- "ULDataReader class" on page 220
- "ULTable class" on page 394
- System.Data.DataSet

ULDataAdapter constructor

Initializes a ULDataAdapter object.

Overload list

Name	Description
"ULDataAdapter() constructor"	Initializes a ULDataAdapter object.
"ULDataAdapter(string, string) constructor"	Initializes a ULDataAdapter object with the specified SELECT statement and connection string.
"ULDataAdapter(string, ULConnection) constructor"	Initializes a ULDataAdapter object with the specified SELECT statement and connection.
"ULDataAdapter(ULCommand) constructor"	Initializes a ULDataAdapter object with the specified SELECT statement.

ULDataAdapter() constructor

Initializes a ULDataAdapter object.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ULDataAdapter()
```

See also

- "ULDataAdapter constructor" on page 199

ULDataAdapter(string, string) constructor

Initializes a ULDataAdapter object with the specified SELECT statement and connection string.

Visual Basic syntax

```
Public Sub New(
    ByVal selectCommandText As String,
    ByVal selectConnectionString As String
)
```

C# syntax

```
public ULDataAdapter(
    string selectCommandText,
    string selectConnectionString
)
```

Parameters

- **selectCommandText** A SELECT statement to be used by the ULDataAdapter.SelectCommand of the ULDataAdapter.

- **selectConnectionString** A connection string for an UltraLite.NET database.

See also

- "ULDataAdapter constructor" on page 199
- "SelectCommand property" on page 203

ULDataAdapter(string, ULConnection) constructor

Initializes a ULDataAdapter object with the specified SELECT statement and connection.

Visual Basic syntax

```
Public Sub New(
    ByVal selectCommandText As String,
    ByVal selectConnection As ULConnection
)
```

C# syntax

```
public ULDataAdapter(
    string selectCommandText,
    ULConnection selectConnection
)
```

Parameters

- **selectCommandText** A SELECT statement to be used by the ULDataAdapter.SelectCommand of the ULDataAdapter.
- **selectConnection** A ULConnection object that defines a connection to a database.

See also

- "ULDataAdapter constructor" on page 199
- "SelectCommand property" on page 203
- "ULConnection class" on page 110

ULDataAdapter(ULCommand) constructor

Initializes a ULDataAdapter object with the specified SELECT statement.

Visual Basic syntax

```
Public Sub New(ByVal selectCommand As ULCommand)
```

C# syntax

```
public ULDataAdapter(ULCommand selectCommand)
```

Parameters

- **selectCommand** A ULCommand object that is used during System.Data.Common.DbDataAdapter.Fill(System.Data.DataSet) to select records from the data source for placement in the System.Data.DataSet.

See also

- "ULDataAdapter constructor" on page 199
- "ULCommand class" on page 64
- System.Data.DataSet

GetFillParameters method

Returns the parameters set by the user when executing a SELECT statement.

Visual Basic syntax

```
Public Shadows Function GetFillParameters() As ULParameter()
```

C# syntax

```
public new ULParameter[] GetFillParameters()
```

Returns

An array of ULParameter objects that contains the parameters set by the user.

Remarks

This is the strongly-typed version of System.Data.Common.DbDataAdapter.GetFillParameters.

See also

- "ULParameter class" on page 301
- System.Data.Common.DbDataAdapter.GetFillParameters

DeleteCommand property

Specifies a ULCommand object that is executed against the database when DbDataAdapter.Update(System.Data.DataSet) is called to delete rows in the database that correspond to deleted rows in the System.Data.DataSet.

Visual Basic syntax

```
Public Shadows Property DeleteCommand As ULCommand
```

C# syntax

```
public new ULCommand DeleteCommand {get;set;}
```


Remarks

A ULCommand object that is executed to delete rows in the database that correspond to deleted rows in the System.Data.DataSet.

When DeleteCommand is assigned to an existing ULCommand object, the ULCommand object is not cloned. The DeleteCommand maintains a reference to the existing ULCommand.

This is the strongly-typed version of System.Data.IDbDataAdapter.DeleteCommand and System.Data.Common.DbDataAdapter.DeleteCommand.

See also

- "ULCommand class" on page 64
- System.Data.DataSet

InsertCommand property

Specifies a ULCommand object that is executed against the database when DbDataAdapter.Update(System.Data.DataSet) is called to insert rows in the database that correspond to inserted rows in the System.Data.DataSet.

Visual Basic syntax

```
Public Shadows Property InsertCommand As ULCommand
```

C# syntax

```
public new ULCommand InsertCommand {get;set;}
```

Remarks

A ULCommand object that is executed to insert rows in the database that correspond to inserted rows in the System.Data.DataSet.

When InsertCommand is assigned to an existing ULCommand object, the ULCommand object is not cloned. The InsertCommand maintains a reference to the existing ULCommand.

This is the strongly-typed version of System.Data.IDbDataAdapter.InsertCommand and System.Data.Common.DbDataAdapter.InsertCommand.

See also

- "ULCommand class" on page 64
- System.Data.DataSet

SelectCommand property

Specifies a ULCommand that is used during System.Data.Common.DbDataAdapter.Fill(System.Data.DataSet) or

System.Data.Common.DbDataAdapter.FillSchema(System.Data.DataSet,System.Data.SchemaType) to obtain a result set from the database for copying into a System.Data.DataSet.

Visual Basic syntax

```
Public Shadows Property SelectCommand As ULCommand
```

C# syntax

```
public new ULCommand SelectCommand {get;set;}
```

Remarks

A ULCommand object that is executed to fill the System.Data.DataSet.

When SelectCommand is assigned to an existing ULCommand object, the ULCommand object is not cloned. The SelectCommand maintains a reference to the existing ULCommand.

If the SelectCommand does not return any rows, no tables are added to the System.Data.DataSet, and no exception is raised. The SELECT statement can also be specified in the ULDataAdapter(ULCommand), ULDataAdapter(String,ULConnection), or ULDataAdapter(String,String) constructors.

This is the strongly-typed version of System.Data.IDbDataAdapter.SelectCommand and System.Data.Common.DbDataAdapter.SelectCommand.

See also

- "ULCommand class" on page 64
- "ULDataAdapter constructor" on page 199
- System.Data.DataSet

TableMappings property

Returns a collection that provides the master mapping between a source table and a System.Data.DataTable

Visual Basic syntax

```
Public ReadOnly Shadows Property TableMappings As
DataTableMappingCollection
```

C# syntax

```
public new DataTableMappingCollection TableMappings {get;}
```

Remarks

A collection of System.Data.Common.DataTableMapping objects providing the master mapping between source tables and System.Data.DataTables. The default value is an empty collection.

When reconciling changes, the ULDataAdapter uses the System.Data.Common.DataTableMappingCollection collection to associate the column names used by the data source with the column names used by the System.Data.DataSet.

This is the strongly-typed version of System.Data.IDataAdapter.TableMappings.

See also

- System.Data.DataTable

UpdateCommand property

Specifies a ULCommand object that is executed against the database when System.Data.Common.DbDataAdapter.Update(System.Data.DataSet) is called to update rows in the database that correspond to updated rows in the System.Data.DataSet.

Visual Basic syntax

```
Public Shadows Property UpdateCommand As ULCommand
```

C# syntax

```
public new ULCommand UpdateCommand {get;set;}
```

Remarks

A ULCommand object that is executed to update rows in the database that correspond to updated rows in the System.Data.DataSet.

When UpdateCommand is assigned to an existing ULCommand object, the ULCommand object is not cloned. The UpdateCommand maintains a reference to the existing ULCommand.

If execution of this command returns rows, these rows may be merged with the System.Data.DataSet depending on how you set the ULCommand.UpdatedRowSource of the ULCommand object.

This is the strongly-typed version of System.Data.IDbDataAdapter.UpdateCommand and System.Data.Common.DbDataAdapter.DeleteCommand.

See also

- "ULCommand class" on page 64
- "UpdatedRowSource property" on page 99
- System.Data.DataSet

RowUpdated event

Occurs during an update after a command is executed against the data source.

Visual Basic syntax

```
Public Event RowUpdated As ULRowUpdatedEventHandler
```

C# syntax

```
public event ULRowUpdatedEventHandler RowUpdated;
```

Remarks

When an attempt to update is made, the event fires.

To process row updated events, you must create a ULRowUpdatedEventHandler delegate and attach it to this event.

See also

- "ULRowUpdatedEventHandler delegate" on page 433

RowUpdating event

Occurs during an update before a command is executed against the data source.

Visual Basic syntax

```
Public Event RowUpdating As ULRowUpdatingEventHandler
```

C# syntax

```
public event ULRowUpdatingEventHandler RowUpdating;
```

Remarks

When an attempt to update is made, the event fires.

To process row updating events, you must create a ULRowUpdatingEventHandler delegate and attach it to this event.

See also

- "ULRowUpdatedEventHandler delegate" on page 433

ULDatabaseManager class

UL Ext: Provides static methods for creating, deleting, and validating databases.

Visual Basic syntax

```
Public NotInheritable Class ULDatabaseManager
```

C# syntax

```
public sealed class ULDatabaseManager
```

Members

All members of ULDatabaseManager class, including all inherited members.

Name	Description
"CreateDatabase method"	Creates a new UltraLite database.
"DropDatabase method"	Deletes the specified database.
"SetActiveSyncListener method"	Specifies the listener object used to process ActiveSync calls from the MobiLink provider for ActiveSync.
"SetServerSyncListener method"	Specifies the listener object used to process the specified server synchronization message.
"SignalSyncIsComplete method"	Signals the MobiLink provider for ActiveSync that an application has completed synchronization.
"ValidateDatabase method"	Performs low level and index validation on a database.
"RuntimeType property"	Specifies the UltraLite.NET runtime type.

Remarks

To use the UltraLite Engine runtime of UltraLite.NET, set ULDatabaseManager.RuntimeType to the appropriate value before using any other UltraLite.NET API.

Example

The following example selects the UltraLite Engine runtime and creates a connection.

```
' Visual Basic
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT
Dim conn As ULConnection = new ULConnection
' The RuntimeType is now locked
```

The following code is the C# language equivalent:

```
// C#
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT;
ULConnection conn = new ULConnection();
// The RuntimeType is now locked
```

CreateDatabase method

Creates a new UltraLite database.

Visual Basic syntax

```
Public Shared Sub CreateDatabase(
    ByVal connString As String,
    ByVal createParms As String
)
```

C# syntax

```
public static void CreateDatabase(string connString, string createParms)
```

Parameters

- **connString** The parameters for identifying a database in the form of a semicolon-separated list of keyword-value pairs.
- **createParms** The parameters used to configure the new database in the form of a semicolon-separated list of keyword-value pairs.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "Open method" on page 136
- "ULConnectionParms class" on page 154
- "ULCreateParms class" on page 179

Example

The following code creates the database \UltraLite\MyDatabase.udb on a Windows Mobile device then opens a connection to it.

```
' Visual Basic
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"
ULConnection.DatabaseManager.CreateDatabase( _
    openParms.ToString(), _
    "" _
)
Dim conn As ULConnection = _
    New ULConnection( openParms.ToString() )
conn.Open()
```

The following code is the C# language equivalent:

```
// C#
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnDevice = ".udb";

ULConnection.DatabaseManager.CreateDatabase(
    openParms.ToString(),
    ""
);
ULConnection conn = new ULConnection( openParms.ToString() );
conn.Open();
```

DropDatabase method

Deletes the specified database.

Visual Basic syntax

```
Public Shared Sub DropDatabase(ByVal connString As String)
```

C# syntax

```
public static void DropDatabase(string connString)
```

Parameters

- **connString** The parameters for identifying a database in the form of a semicolon-separated list of keyword-value pairs.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

You cannot drop a database that has open connections.

See also

- "Open method" on page 136
- "ULConnectionParms class" on page 154

Example

The following code creates the database \UltraLite\MyDatabase.udb on a Windows Mobile device then opens a connection to it.

```
' Visual Basic
Dim connParms As ULConnectionParms = New ULConnectionParms
connParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"
ULConnection.DatabaseManager.DropDatabase( _
    connParms.ToString() _
)
```

The following code is the C# language equivalent:

```
// C#
ULConnectionParms connParms = new ULConnectionParms();
connParms.DatabaseOnDevice = ".udb";
ULConnection.DatabaseManager.DropDatabase(
    connParms.ToString()
);
ULConnection conn = new ULConnection( openParms.ToString() );
conn.Open();
```

SetActiveSyncListener method

Specifies the listener object used to process ActiveSync calls from the MobiLink provider for ActiveSync.

Visual Basic syntax

```
Public Shared Sub SetActiveSyncListener(
    ByVal appClassName As String,
```

```
    ByVal listener As ULActiveSyncListener
)
```

C# syntax

```
public static void SetActiveSyncListener(
    string appClassName,
    ULActiveSyncListener listener
)
```

Parameters

- **appClassName** The unique class name for the application. This is the class name used when the application is registered for use with ActiveSync.
- **listener** The ULActiveSyncListener object. Use null (Nothing in Visual Basic) to remove the previous listener.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The parameter *appClassName* is the unique identifier used to identify the application. The application can only use one *appClassName* at a time. While a listener is registered with a particular *appClassName* , calls to SetServerSyncListener or SetActiveSyncListener with a different *appClassName* fail.

To remove the ActiveSync listener, call SetActiveSyncListener with a null reference (Nothing in Visual Basic) as the *listener* parameter.

To remove all listeners, call SetServerSyncListener with a null reference (Nothing in Visual Basic) for all parameters.

Applications should remove all listeners prior to exiting.

See also

- "SetServerSyncListener method" on page 210
- "SetActiveSyncListener method" on page 209
- "ULActiveSyncListener interface" on page 37
- "ActiveSyncInvoked method" on page 37

SetServerSyncListener method

Specifies the listener object used to process the specified server synchronization message.

Visual Basic syntax

```
Public Shared Sub SetServerSyncListener(
    ByVal messageName As String,
    ByVal appClassName As String,
    ByVal listener As ULServerSyncListener
)
```


C# syntax

```
public static void SetServerSyncListener(
    string messageName,
    string appClassName,
    ULServerSyncListener listener
)
```

Parameters

- **messageName** The name of the message.
- **appClassName** The unique class name for the application. This is a unique identifier used to identify the application.
- **listener** The ULServerSyncListener object. Use null (Nothing in Visual Basic) to remove the previous listener.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The parameter *appClassName* is the unique identifier used to identify the application. The application may only use one *appClassName* at a time. While a listener is registered with a particular *appClassName* , calls to SetServerSyncListener or SetActiveSyncListener with a different *appClassName* fail.

To remove the listener for a particular message, call SetServerSyncListener with a null reference (Nothing in Visual Basic) as the *listener* parameter.

To remove all listeners, call SetServerSyncListener with a null reference (Nothing in Visual Basic) for all parameters.

Applications should remove all listeners before exiting.

See also

- "SetServerSyncListener method" on page 210
- "SetActiveSyncListener method" on page 209
- "ServerSyncInvoked method" on page 367

SignalSyncIsComplete method

Signals the MobiLink provider for ActiveSync that an application has completed synchronization.

Visual Basic syntax

```
Public Shared Sub SignalSyncIsComplete()
```

C# syntax

```
public static void SignalSyncIsComplete()
```

See also

- "SignalSyncIsComplete method" on page 211
- "ActiveSyncInvoked method" on page 37

ValidateDatabase method

Performs low level and index validation on a database.

Visual Basic syntax

```
Public Shared Sub ValidateDatabase(
    ByVal start_parms As String,
    ByVal how As ULDBValid
)
```

C# syntax

```
public static void ValidateDatabase(string start_parms, ULDBValid how)
```

Parameters

- **start_parms** The parameters for identifying a database in the form of a semicolon-separated list of keyword-value pairs.
- **how** Describes how to validate the database.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ValidateDatabase method" on page 144
- "ULDBValid enumeration" on page 437
- "ULConnectionParms class" on page 154

Example

The following code validates indexes for the database \UltraLite\MyDatabase.udb under Windows Mobile:

```
' Visual Basic
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"
ULConnection.DatabaseManager.ValidateDatabase( _
    openParms.ToString(), iAnywhere.Data.UltraLite.ULVF_INDEX )
```

The following code is the C# language equivalent:

```
// C#
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnDevice = ".udb";
ULConnection.DatabaseManager.ValidateDatabase(
    openParms.ToString(), iAnywhere.Data.UltraLite.ULVF_INDEX );
```


RuntimeType property

Specifies the UltraLite.NET runtime type.

Visual Basic syntax

```
Public Shared Property RuntimeType As ULRuntimeType
```

C# syntax

```
public ULRuntimeType RuntimeType {get;set;}
```

Remarks

The runtime type must be selected before using any other UltraLite.NET API.

A ULRuntimeType value identifying the type of the unmanaged UltraLite.NET runtime.

See also

- "ULRuntimeType enumeration" on page 441

Example

The following example selects the UltraLite Engine runtime and creates a connection.

```
' Visual Basic
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT
Dim conn As ULConnection = new ULConnection
' The RuntimeType is now locked
```

The following code is the C# language equivalent:

```
// C#
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT;
ULConnection conn = new ULConnection();
// The RuntimeType is now locked
```

ULDatabaseSchema class

UL Ext: Represents the schema of an UltraLite.NET database.

Visual Basic syntax

```
Public NotInheritable Class ULDatabaseSchema
```

C# syntax

```
public sealed class ULDatabaseSchema
```

Members

All members of ULDatabaseSchema class, including all inherited members.

Name	Description
"GetDatabaseProperty method"	Returns the value of the specified database property.
"GetPublicationName method"	Returns the name of the publication identified by the specified publication ID.
"GetTableName method"	Returns the name of the table identified by the specified table ID.
"SetDatabaseOption method"	Sets the value for the specified database option.
"IsCaseSensitive property"	Checks whether the database is case sensitive.
"IsOpen property"	Whether the database schema is open.
"PublicationCount property"	The number of publications in the database.
"TableCount property"	The number of tables in the database.

Remarks

There is no constructor for this class. A ULDatabaseSchema object is attached to a connection as its ULConnection.Schema and is only valid while that connection is open.

See also

- "ULDatabaseSchema class" on page 213
- "Schema property" on page 149

GetDatabaseProperty method

Returns the value of the specified database property.

Visual Basic syntax

```
Public Function GetDatabaseProperty(ByVal name As String) As String
```

C# syntax

```
public string GetDatabaseProperty(string name)
```

Parameters

- **name** The name of the database property whose value you want to obtain. Property names are case insensitive.

Returns

The value of the property as a string.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Recognized properties are:

Property	Description
CaseSensitive	The status of the case sensitivity feature. Returns ON if the database is case sensitive. Otherwise, it returns OFF. Database case sensitivity affects how indexes on tables and result sets are sorted. Case sensitivity does not affect how a connection's ULConnectionParms.UserID and ULConnectionParms.Password are verified. User IDs are always case insensitive and passwords are always case sensitive.
CharSet	The character set of the database.
ChecksumLevel	The level of database page checksums enabled for the database.
Collation	The name of the database's collation sequence.
ConnCount	The number of connections to the database.
date_format	The date format used for string conversions by the database. This format is not necessarily the same as the one used by System.DateTime.
date_order	The date order used for string conversions by the database.
Encryption	The type of encryption applied to the database. Returns None, Simple, AES, or AES_FIPS.
File	The file name of the database.
global_database_id	The value of the global_database_id option used for global autoincrement columns.
isolation_level	The value of the isolation_level option used for controlling the degree to which the operations in one transaction are visible to the operations in other concurrent transactions. This value is set on a per connection basis.
MaxHashSize	The default maximum number of bytes to use for index hashing. This property can be set on a per-index basis.
ml_remote_id	The value of the ml_remote_id option used for identifying the database during synchronization.
Name	The name of the database (DBN).

Property	Description
nearest_century	The nearest century used for string conversions by the database.
PageSize	The page size of the database, in bytes.
precision	The floating-point precision used for string conversions by the database.
scale	The minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION during string conversions by the database.
time_format	The time format used for string conversions by the database. This format is not necessarily the same as the one used by System.TimeSpan.
timestamp_format	The timestamp format used for string conversions by the database. This format is not necessarily the same as the one used by System.DateTime.
timestamp_increment	The minimum difference between two unique timestamps, in microseconds (1,000,000th of a second).

See also

- "SetDatabaseOption method" on page 217
- "UserID property" on page 163
- "Password property" on page 162
- System.TimeSpan

GetPublicationName method

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

Visual Basic syntax

```
Public Function GetPublicationName(ByVal pubID As Integer) As String
```

C# syntax

```
public string GetPublicationName(int pubID)
```

Parameters

- **pubID** The ID of the publication. The value must be in the range [1,PublicationCount].

Returns

The publication name as a string.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

> **Note**
> Publication IDs and counts may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached IDs and counts after a schema upgrade.

See also

- "PublicationCount property" on page 219

GetTableName method

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

Visual Basic syntax

```
Public Function GetTableName(ByVal tableID As Integer) As String
```

C# syntax

```
public string GetTableName(int tableID)
```

Parameters

- **tableID** The ID of the table. The value must be in range [1,TableCount].

Returns

The table name as a string.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Table IDs may change during a schema upgrade. To correctly identify a table, access it by name or refresh the cached IDs after a schema upgrade.

See also

- "TableCount property" on page 220

SetDatabaseOption method

Sets the value for the specified database option.

Visual Basic syntax

```
Public Sub SetDatabaseOption(
    ByVal name As String,
```

```
        ByVal value As String
    )
```

C# syntax

```
public void SetDatabaseOption(string name, string value)
```

Parameters

- **name** The name of the database option. Option names are case insensitive.
- **value** The new value for the option.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Setting a database option results in a commit being performed.

Recognized options are:

Option	Description
global_database_id	The value used for global autoincrement columns. The value must be in the range [0,System.UInt32.MaxValue]. The default is ULConnection.INVALID_DATABASE_ID (used to indicate that the database ID has not been set for the current database).
isolation_level	The value used to control the degree to which the operations in one transaction are visible to the operations in other concurrent transactions. The value must be one of "read_uncommitted" or "read_committed". The default is "read_committed". Setting the isolation_level on a connection to "read_uncommited" is equivalent to wrapping all operations on that connection with BeginTransaction(System.Data.IsolationLevel.ReadUncommitted) and Commit() calls. Similarly, "read_committed" is equivalent to System.Data.IsolationLevel.ReadCommitted. SetDatabaseOption() should not be used to set the current transaction's isolation level; use BeginTransaction(IsolationLevel) instead. UltraLite's definition of each isolation level is slightly different than ADO.NET's documentation of IsolationLevel. For more information, see "UltraLite isolation levels" [*UltraLite - Database Management and Reference*]. This value is set on a per connection basis.
ml_remote_id	The value used for identifying the database during synchronization. Use a null reference (Nothing in Visual Basic) as the value to remove the ml_remote_id option from the database.

See also

- "GetDatabaseProperty method" on page 214
- "INVALID_DATABASE_ID field" on page 153
- System.UInt32.MaxValue

IsCaseSensitive property

Checks whether the database is case sensitive.

Visual Basic syntax

```
Public ReadOnly Property IsCaseSensitive As Boolean
```

C# syntax

```
public bool IsCaseSensitive {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

True if the database is case sensitive, and false if the database is case insensitive.

Database case sensitivity affects how indexes on tables and result sets are sorted. Case sensitivity also affects how ULConnectionParms.UserID and ULConnectionParms.Password are verified.

See also

- "GetDatabaseProperty method" on page 214
- "UserID property" on page 163
- "Password property" on page 162

IsOpen property

Whether the database schema is open.

Visual Basic syntax

```
Public ReadOnly Property IsOpen As Boolean
```

C# syntax

```
public bool IsOpen {get;}
```

Remarks

True if this database schema is currently open, false if this database schema is currently closed.

A ULDatabaseSchema object is open only if the connection it is attached to is open.

PublicationCount property

The number of publications in the database.

Visual Basic syntax

```
Public ReadOnly Property PublicationCount As Integer
```

C# syntax

```
public int PublicationCount {get;}
```

Remarks

The number of publications in the database.

Publication IDs range from 1 to PublicationCount, inclusively.

> **Note**
> Publication IDs and counts may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached IDs and counts after a schema upgrade.

See also

- "GetPublicationName method" on page 216

TableCount property

The number of tables in the database.

Visual Basic syntax

```
Public ReadOnly Property TableCount As Integer
```

C# syntax

```
public int TableCount {get;}
```

Remarks

The number of tables in the database.

Table IDs range from 1 to TableCount, inclusively.

> **Note**
> Table IDs and counts may change during a schema upgrade. To correctly identify a table, access it by name or refresh the cached IDs and counts after a schema upgrade.

ULDataReader class

Represents a read-only bi-directional cursor in an UltraLite database.

Visual Basic syntax

```
Public Class ULDataReader
    Inherits System.Data.Common.DbDataReader
    Implements System.ComponentModel.IListSource
```

C# syntax

```
public class ULDataReader :
    System.Data.Common.DbDataReader,
    System.ComponentModel.IListSource
```

Base classes

- System.Data.Common.DbDataReader
- System.ComponentModel.IListSource

Derived classes

- "ULResultSet class" on page 334

Members

All members of ULDataReader class, including all inherited members.

Name	Description
"Close method"	Closes the cursor.
Dispose method (Inherited from System.Data.Common.DbDataReader)	Releases all resources used by the current instance of the System.Data.Common.DbDataReader class.
"GetBoolean method"	Returns the value for the specified column as a System.Boolean.
"GetByte method"	Returns the value for the specified column as an unsigned 8-bit value (System.Byte).
"GetBytes method"	**UL Ext:** Returns the value for the specified column as an array of System.Bytes.
"GetChar method"	This method is not supported in UltraLite.NET.
"GetChars method"	Copies a subset of the value for the specified ULDbType.LongVarchar column, beginning at the specified offset, to the specified offset of the destination System.Char array.

Name	Description
GetData method (Inherited from System.Data.Common.DbDataReader)	Returns a System.Data.Common.DbDataReader object for the requested column ordinal.
"GetDataTypeName method"	Returns the name of the specified column's provider data type.
"GetDateTime method"	Returns the value for the specified column as a System.DateTime with millisecond accuracy.
GetDbDataReader method (Inherited from System.Data.Common.DbDataReader)	Returns a System.Data.Common.DbDataReader object for the requested column ordinal that can be overridden with a provider-specific implementation.
"GetDecimal method"	Returns the value for the specified column as a System.Decimal.
"GetDouble method"	Returns the value for the specified column as a System.Double.
"GetEnumerator method"	Returns an System.Collections.IEnumerator that iterates through the ULDataReader.
"GetFieldType method"	Returns the System.Type most appropriate for the specified column.
"GetFloat method"	Returns the value for the specified column as a System.Single.
"GetGuid method"	Returns the value for the specified column as a UUID (System.Guid).
"GetInt16 method"	Returns the value for the specified column as a System.Int16.
"GetInt32 method"	Returns the value for the specified column as an Int32.
"GetInt64 method"	Returns the value for the specified column as an Int64.

Name	Description
"GetName method"	Returns the name of the specified column.
"GetOrdinal method"	Returns the column ID of the named column.
GetProviderSpecificFieldType method (Inherited from System.Data.Common.DbDataReader)	Returns the provider-specific field type of the specified column.
GetProviderSpecificValue method (Inherited from System.Data.Common.DbDataReader)	Gets the value of the specified column as an instance of System.Object.
GetProviderSpecificValues method (Inherited from System.Data.Common.DbDataReader)	Gets all provider-specific attribute columns in the collection for the current row.
"GetRowCount method"	**UL Ext:** Returns the number of rows in the cursor, within threshold.
"GetSchemaTable method"	Returns a System.Data.DataTable that describes the column metadata of the ULDataReader.
"GetString method"	Returns the value for the specified column as a System.String.
"GetTimeSpan method"	Returns the value for the specified column as a System.TimeSpan with millisecond accuracy.
"GetUInt16 method"	Returns the value for the specified column as a System.UInt16.
"GetUInt32 method"	Returns the value for the specified column as a UInt32.
"GetUInt64 method"	Returns the value for the specified column as a System.UInt64.
"GetValue method"	Returns the value of the specified column in its native format.
"GetValues method"	Returns all the column values for the current row.

Name	Description
“IsDBNull method”	Checks whether the value from the specified column is NULL.
“MoveAfterLast method”	**UL Ext:** Positions the cursor to after the last row of the cursor.
“MoveBeforeFirst method”	**UL Ext:** Positions the cursor to before the first row of the cursor.
“MoveFirst method”	**UL Ext:** Positions the cursor to the first row of the cursor.
“MoveLast method”	**UL Ext:** Positions the cursor to the last row of the cursor.
“MoveNext method”	**UL Ext:** Positions the cursor to the next row or after the last row if the cursor was already on the last row.
“MovePrevious method”	**UL Ext:** Positions the cursor to the previous row or before the first row.
“MoveRelative method”	**UL Ext:** Positions the cursor relative to the current row.
“NextResult method”	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
“Read method”	Positions the cursor to the next row, or after the last row if the cursor was already on the last row.
“Depth property”	Returns the depth of nesting for the current row.
“FieldCount property”	Returns the number of columns in the cursor.
“HasRows property”	Checks whether the ULDataReader has one or more rows.
“IsBOF property”	**UL Ext:** Checks whether the current row position is before the first row.

Name	Description
"IsClosed property"	Checks whether the cursor is currently open.
"IsEOF property"	**UL Ext:** Checks whether the current row position is after the last row.
"RecordsAffected property"	Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.
"RowCount property"	**UL Ext:** Returns the number of rows in the cursor.
"Schema property"	**UL Ext:** Holds the schema of this cursor.
"this property"	Returns the value of the specified column in its native format.
VisibleFieldCount property (Inherited from System.Data.Common.DbDataReader)	Gets the number of fields in the System.Data.Common.DbDataReader that are not hidden.

Remarks

Cursors are sets of rows from either a table or the result set from a query.

There is no constructor for ULDataReader. To get a ULDataReader object, execute a ULCommand:

```
' Visual Basic
Dim cmd As ULCommand = new ULCommand( _
    "SELECT emp_id FROM employee", conn _
)
Dim reader As ULDataReader = cmd.ExecuteReader()
```

The following code is the C# language equivalent:

```
// C#
ULCommand cmd = new ULCommand(
    "SELECT emp_id FROM employee", conn
);
ULDataReader reader = cmd.ExecuteReader();
```

UL Ext: The ADO.NET standard only requires forward-only motion through the result set, but ULDataReader is bi-directional. ULDataReader's Move methods provide you with full flexibility when moving through results.

ULDataReader is a read-only result set. If you need a more flexible object to manipulate results, use a ULCommand.ExecuteResultSet(), ULCommand.ExecuteTable(), or a ULDataAdapter. ULDataReader retrieves rows as needed, whereas ULDataAdapter must retrieve all rows of a result set before you can

carry out any action on the object. For large result sets, this difference gives the ULDataReader a much faster response time.

UL Ext: All columns of a ULDataReader may be retrieved using GetString.

See also

- "ULCommand class" on page 64
- "ExecuteResultSet method" on page 87
- "ExecuteTable method" on page 91
- "ULDataAdapter class" on page 196
- "GetString method" on page 243
- System.Data.Common.DbDataReader

Close method

Closes the cursor.

Visual Basic syntax

```
Public Overrides Sub Close()
```

C# syntax

```
public override void Close()
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

It is not an error to close a cursor that is already closed.

GetBoolean method

Returns the value for the specified column as a System.Boolean.

Visual Basic syntax

```
Public Overrides Function GetBoolean(ByVal colID As Integer) As Boolean
```

C# syntax

```
public override bool GetBoolean(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as a System.Boolean.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.Boolean

GetByte method

Returns the value for the specified column as an unsigned 8-bit value (System.Byte).

Visual Basic syntax

```
Public Overrides Function GetByte(ByVal colID As Integer) As Byte
```

C# syntax

```
public override byte GetByte(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as a System.Byte.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.Byte

GetBytes method

UL Ext: Returns the value for the specified column as an array of System.Bytes.

Overload list

Name	Description
"GetBytes(int) method"	**UL Ext:** Returns the value for the specified column as an array of System.Bytes.
"GetBytes(int, long, byte[], int, int) method"	Copies a subset of the value for the specified ULDbType.LongBinary column, beginning at the specified offset, to the specified offset of the destination System.Byte array.

GetBytes(int) method

UL Ext: Returns the value for the specified column as an array of System.Bytes.

Visual Basic syntax

```
Public Function GetBytes(ByVal colID As Integer) As Byte()
```

C# syntax

```
public byte[] GetBytes(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as an array of System.Bytes.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Only valid for columns of type ULDbType.Binary, ULDbType.LongBinary, or ULDbType.UniqueIdentifier.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- System.Byte

GetBytes(int, long, byte[], int, int) method

Copies a subset of the value for the specified ULDbType.LongBinary column, beginning at the specified offset, to the specified offset of the destination System.Byte array.

Visual Basic syntax

```
Public Overrides Function GetBytes(
    ByVal colID As Integer,
    ByVal srcOffset As Long,
    ByVal dst As Byte(),
    ByVal dstOffset As Integer,
    ByVal count As Integer
) As Long
```

C# syntax

```
public override long GetBytes(
    int colID,
    long srcOffset,
    byte[] dst,
    int dstOffset,
    int count
)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **srcOffset** The start position in the column value. Zero is the beginning of the value.
- **dst** The destination array.
- **dstOffset** The start position in the destination array.
- **count** The number of bytes to be copied.

Returns

The actual number of bytes copied.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

If you pass a *dst* buffer that is a null reference (Nothing in Visual Basic), GetBytes returns the length of the field in bytes.

The bytes at position *srcOffset* through *srcOffset* +*count* -1 of the value are copied into positions *dstOffset* through *dstOffset* +*count* -1, respectively, of the destination array. If the end of the value is encountered before *count* bytes are copied, the remainder of the destination array is left unchanged.

If any of the following is true, a ULException with code ULSQLCode.SQLE_INVALID_PARAMETER is thrown and the destination is not modified:

- *srcOffset* is negative.
- *dstOffset* is negative.

- *count* is negative.
- *dstOffset* +*count* is greater than the *dst* length.

For other errors, a ULException with the appropriate error code is thrown.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "GetBytes method" on page 227
- "ULException class" on page 257
- "FieldCount property" on page 252
- System.Byte

GetChar method

This method is not supported in UltraLite.NET.

Visual Basic syntax

```
Public Overrides Function GetChar(ByVal colID As Integer) As Char
```

C# syntax

```
public override char GetChar(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

This method is not supported in UltraLite.NET.

Exceptions

- **"ULException class"** This method is not supported in UltraLite.NET.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "GetString method" on page 243
- "FieldCount property" on page 252

GetChars method

Copies a subset of the value for the specified ULDbType.LongVarchar column, beginning at the specified offset, to the specified offset of the destination System.Char array.

Visual Basic syntax

```
Public Overrides Function GetChars(
    ByVal colID As Integer,
    ByVal srcOffset As Long,
    ByVal dst As Char(),
    ByVal dstOffset As Integer,
    ByVal count As Integer
) As Long
```

C# syntax

```
public override long GetChars(
    int colID,
    long srcOffset,
    char[] dst,
    int dstOffset,
    int count
)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **srcOffset** The start position in the column value. Zero is the beginning of the value.
- **dst** The destination array.
- **dstOffset** The start position in the destination array.
- **count** The number of characters to be copied.

Returns

The actual number of characters copied.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

If you pass a *dst* buffer that is a null reference (Nothing in Visual Basic), GetChars returns the length of the field in characters.

The characters at position *srcOffset* through *srcOffset* +*count* -1 of the value are copied into positions *dstOffset* through *dstOffset* +*count* -1, respectively, of the destination array. If the end of the value is encountered before *count* characters are copied, the remainder of the destination array is left unchanged.

If any of the following is true, a ULException with code ULSQLCode.SQLE_INVALID_PARAMETER is thrown and the destination is not modified:

- *srcOffset* is negative.
- *dstOffset* is negative.

- *count* is negative.
- *dstOffset* +*count* is greater than the *dst* length.

For other errors, a ULException with the appropriate error code is thrown.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "ULException class" on page 257
- "FieldCount property" on page 252
- System.Char

GetDataTypeName method

Returns the name of the specified column's provider data type.

Visual Basic syntax

```
Public Overrides Function GetDataTypeName(
    ByVal colID As Integer
) As String
```

C# syntax

```
public override string GetDataTypeName(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

A string corresponding to the column's ULDbType.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetColumnULDbType method" on page 194
- "FieldCount property" on page 252
- "ULDbType enumeration" on page 437

GetDateTime method

Returns the value for the specified column as a System.DateTime with millisecond accuracy.

Visual Basic syntax

```
Public Overrides Function GetDateTime(ByVal colID As Integer) As Date
```

C# syntax

```
public override DateTime GetDateTime(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as a System.DateTime.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.DateTime

GetDecimal method

Returns the value for the specified column as a System.Decimal.

Visual Basic syntax

```
Public Overrides Function GetDecimal(ByVal colID As Integer) As Decimal
```

C# syntax

```
public override decimal GetDecimal(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as a System.Decimal.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.Decimal

GetDouble method

Returns the value for the specified column as a System.Double.

Visual Basic syntax

```
Public Overrides Function GetDouble(ByVal colID As Integer) As Double
```

C# syntax

```
public override double GetDouble(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as a System.Double.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.Double

GetEnumerator method

Returns an System.Collections.IEnumerator that iterates through the ULDataReader.

Visual Basic syntax

```
Public Overrides Function GetEnumerator()
    As System.Collections.IEnumerator
```

C# syntax

```
public override IEnumerator GetEnumerator()
```


Returns

A System.Collections.IEnumerator for the ULDataReader.

See also

- System.Collections.IEnumerator

GetFieldType method

Returns the System.Type most appropriate for the specified column.

Visual Basic syntax

```
Public Overrides Function GetFieldType(ByVal colID As Integer) As Type
```

C# syntax

```
public override Type GetFieldType(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

A System.Type value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetDataTypeName method" on page 232
- "GetColumnULDbType method" on page 194
- "FieldCount property" on page 252
- System.Type

GetFloat method

Returns the value for the specified column as a System.Single.

Visual Basic syntax

```
Public Overrides Function GetFloat(ByVal colID As Integer) As Single
```

C# syntax

```
public override float GetFloat(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as a System.Single.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.Single

GetGuid method

Returns the value for the specified column as a UUID (System.Guid).

Visual Basic syntax

```
Public Overrides Function GetGuid(ByVal colID As Integer) As Guid
```

C# syntax

```
public override Guid GetGuid(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as a Guid.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

This method is only valid for columns of type ULDbType.UniqueIdentifier or for columns of type ULDbType.Binary with length 16.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "GetColumnULDbType method" on page 194
- "GetColumnSize method" on page 193
- "FieldCount property" on page 252
- System.Guid

GetInt16 method

Returns the value for the specified column as a System.Int16.

Visual Basic syntax

```
Public Overrides Function GetInt16(ByVal colID As Integer) As Short
```

C# syntax

```
public override short GetInt16(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as an System.Int16.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.Int16

GetInt32 method

Returns the value for the specified column as an Int32.

Visual Basic syntax

```
Public Overrides Function GetInt32(ByVal colID As Integer) As Integer
```

C# syntax

```
public override int GetInt32(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as an Int32.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.Int32

GetInt64 method

Returns the value for the specified column as an Int64.

Visual Basic syntax

```
Public Overrides Function GetInt64(ByVal colID As Integer) As Long
```

C# syntax

```
public override long GetInt64(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as an Int64.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.Int64

GetName method

Returns the name of the specified column.

Visual Basic syntax

```
Public Overrides Function GetName(ByVal colID As Integer) As String
```

C# syntax

```
public override string GetName(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Note that in result sets, not all columns have names and not all column names are unique. If you are not using aliases, the name of a non-computed column is prefixed with the name of the table the column is from. For example, MyTable.ID is the name of the only column in the result set for the query "SELECT ID FROM MyTable".

This method is identical to the ULCursorSchema.GetColumnName method.

See also

- "FieldCount property" on page 252
- "GetSchemaTable method" on page 241
- "GetColumnName method" on page 191
- "FieldCount property" on page 252

GetOrdinal method

Returns the column ID of the named column.

Visual Basic syntax

```
Public Overrides Function GetOrdinal(
    ByVal columnName As String
) As Integer
```

C# syntax

```
public override int GetOrdinal(string columnName)
```

Parameters

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

Returns

The column ID of the named column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Column IDs range from 0 to ULDataReader.FieldCount-1, inclusively.

Note that in result sets, not all columns have names and not all column names are unique. If you are not using aliases, the name of a non-computed column is prefixed with the name of the table the column is from. For example, MyTable.ID is the name of the only column in the result set for the query "SELECT ID FROM MyTable".

Column IDs and counts may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

This method is identical to the ULCursorSchema.GetColumnID method.

See also

- "GetSchemaTable method" on page 241
- "FieldCount property" on page 252
- "GetColumnID method" on page 190

GetRowCount method

UL Ext: Returns the number of rows in the cursor, within threshold.

Visual Basic syntax

```
Public Function GetRowCount(ByVal threshold As Integer) As Integer
```

C# syntax

```
public int GetRowCount(int threshold)
```

Parameters

- **threshold** Threshold limit for row count.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The number of rows in the cursor.

The property RowCount is expensive with complex queries, as it requires passing through the cursor rows. By using GetRowCount(threshold), the caller can determine if there are at least threshold rows. If the number of rows is below the threshold, that number is returned; otherwise, threshold is returned. This can be called again with a higher threshold.

If threshold is 0, returns the RowCount property.

See also

- "RowCount property" on page 255

GetSchemaTable method

Returns a System.Data.DataTable that describes the column metadata of the ULDataReader.

Visual Basic syntax

```
Public Overrides Function GetSchemaTable() As DataTable
```

C# syntax

```
public override DataTable GetSchemaTable()
```

Returns

A System.Data.DataTable describing the schema of each column in the ULDataReader.

Remarks

The GetSchemaTable method returns metadata about each column in the following order:

DataTable column	Description
ColumnName	The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned. Note that in result sets, not all columns have names and not all column names are unique.
ColumnOrdinal	The ID of the column. The value is in the range [0,FieldCount-1].
ColumnSize	For sized columns, the maximum length of a value in the column. For other columns, this is the size in bytes of the data type.

DataTable column	Description
NumericPrecision	The precision of a numeric column (ProviderType ULDbType.Decimal or ULDbType.Numeric) or DBNull if the column is not numeric.
NumericScale	The scale of a numeric column (ProviderType ULDbType.Decimal or ULDbType.Numeric) or DBNull if the column is not numeric.
IsUnique	True if the column is a non-computed unique column in the table (BaseTableName) it is taken from.
IsKey	True if the column is one of a set of columns in the result set that taken together from a unique key for the result set. The set of columns with IsKey set to true does not need to be the minimal set that uniquely identifies a row in the result set.
BaseCatalogName	The name of the catalog in the database that contains the column. For UltraLite.NET, this value is always DBNull.
BaseColumnName	The original name of the column in the table BaseTableName of the database or DBNull if the column is computed or if this information cannot be determined.
BaseSchemaName	The name of the schema in the database that contains the column. For UltraLite.NET, this value is always DBNull.
BaseTableName	The name of the table in the database that contains the column, or DBNull if column is computed or if this information cannot be determined.
DataType	The .NET data type that is most appropriate for this type of column.
AllowDBNull	True if the column is nullable, false if the column is not nullable or if this information cannot be determined.
ProviderType	The ULDbType of the column.
IsIdentity	True if the column is an identity column, false if it is not an identity column. For UltraLite.NET, this value is always false.
IsAutoIncrement	True if the column is an autoincrement or global autoincrement column, false otherwise (or if this information cannot be determined).
IsRowVersion	True if the column contains a persistent row identifier that cannot be written to, and has no meaningful value except to identity the row. For UltraLite.NET, this value is always false.
IsLong	True if the column is a ULDbType.LongVarchar or a ULDbType.LongBinary column, false otherwise.
IsReadOnly	True if the column is read-only, false if the column is modifiable or if its access cannot be determined.

DataTable column	Description
IsAliased	True if the column name is an alias, false if it is not an alias.
IsExpression	True if the column is an expression, false if it is a column value.

See also

- "Schema property" on page 255
- "FieldCount property" on page 252
- "ULDbType enumeration" on page 437
- System.Data.DataTable

GetString method

Returns the value for the specified column as a System.String.

Visual Basic syntax

```
Public Overrides Function GetString(ByVal colID As Integer) As String
```

C# syntax

```
public override String GetString(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as a System.String.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.String

GetTimeSpan method

Returns the value for the specified column as a System.TimeSpan with millisecond accuracy.

Visual Basic syntax

```
Public Function GetTimeSpan(ByVal colID As Integer) As TimeSpan
```

C# syntax

```
public TimeSpan GetTimeSpan(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as a System.TimeSpan.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.TimeSpan

GetUInt16 method

Returns the value for the specified column as a System.UInt16.

Visual Basic syntax

```
Public Function GetUInt16(ByVal colID As Integer) As UShort
```

C# syntax

```
public ushort GetUInt16(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as an System.UInt16.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.UInt16

GetUInt32 method

Returns the value for the specified column as a UInt32.

Visual Basic syntax

```
Public Function GetUInt32(ByVal colID As Integer) As UInteger
```

C# syntax

```
public uint GetUInt32(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as an UInt32.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.UInt32

GetUInt64 method

Returns the value for the specified column as a System.UInt64.

Visual Basic syntax

```
Public Function GetUInt64(ByVal colID As Integer) As ULong
```

C# syntax

```
public ulong GetUInt64(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as a System.UInt64

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252
- System.UInt64

GetValue method

Returns the value of the specified column in its native format.

Visual Basic syntax

```
Public Overrides Function GetValue(ByVal colID As Integer) As Object
```

C# syntax

```
public override object GetValue(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as the .NET type most appropriate for the column or DBNull if column is NULL.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

This method is identical in functionality to the ULDataReader.this[int].

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252

GetValues method

Returns all the column values for the current row.

Visual Basic syntax

```
Public Overrides Function GetValues(ByVal values As Object()) As Integer
```

C# syntax

```
public override int GetValues(object[] values)
```

Parameters

- **values** The array of System.Objects to hold the entire row.

Returns

The number of column values retrieved. If the length of the array is greater than the number of columns (ULDataReader.FieldCount), only FieldCount items are retrieved and the rest of the array is left unchanged.

Exceptions

- **ArgumentNullException** The *values* array is NULL or has zero length.
- **"ULException class"** A SQL error occurred.

Remarks

For most applications, the GetValues method provides an efficient means for retrieving all columns, rather than retrieving each column individually.

You can pass an System.Object array that contains fewer than the number of columns contained in the resulting row. Only the amount of data the System.Object array holds is copied to the array. You can also pass an System.Object array whose length is more than the number of columns contained in the resulting row.

This method returns DBNull for NULL database columns. For other columns, it returns the value of the column in its native format.

See also

- "FieldCount property" on page 252
- "GetFieldType method" on page 235
- "GetValue method" on page 246
- System.Object

IsDBNull method

Checks whether the value from the specified column is NULL.

Visual Basic syntax

```
Public Overrides Function IsDBNull(ByVal colID As Integer) As Boolean
```

C# syntax

```
public override bool IsDBNull(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

True if value is NULL, false if value is not NULL.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "FieldCount property" on page 252

MoveAfterLast method

UL Ext: Positions the cursor to after the last row of the cursor.

Visual Basic syntax

```
Public Sub MoveAfterLast()
```

C# syntax

```
public void MoveAfterLast()
```

Exceptions

- **"ULException class"** A SQL error occurred.

MoveBeforeFirst method

UL Ext: Positions the cursor to before the first row of the cursor.

Visual Basic syntax

```
Public Sub MoveBeforeFirst()
```

C# syntax

```
public void MoveBeforeFirst()
```


Exceptions

- **"ULException class"** A SQL error occurred.

MoveFirst method

UL Ext: Positions the cursor to the first row of the cursor.

Visual Basic syntax

```
Public Function MoveFirst() As Boolean
```

C# syntax

```
public bool MoveFirst()
```

Returns

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

Exceptions

- **"ULException class"** A SQL error occurred.

MoveLast method

UL Ext: Positions the cursor to the last row of the cursor.

Visual Basic syntax

```
Public Function MoveLast() As Boolean
```

C# syntax

```
public bool MoveLast()
```

Returns

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

Exceptions

- **"ULException class"** A SQL error occurred.

MoveNext method

UL Ext: Positions the cursor to the next row or after the last row if the cursor was already on the last row.

Visual Basic syntax

```
Public Function MoveNext() As Boolean
```

C# syntax

```
public bool MoveNext()
```

Returns

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

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

This method is identical to the ULDataReader.Read method.

See also

- "Read method" on page 251

MovePrevious method

UL Ext: Positions the cursor to the previous row or before the first row.

Visual Basic syntax

```
Public Function MovePrevious() As Boolean
```

C# syntax

```
public bool MovePrevious()
```

Returns

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

Exceptions

- **"ULException class"** A SQL error occurred.

MoveRelative method

UL Ext: Positions the cursor relative to the current row.

Visual Basic syntax

```
Public Function MoveRelative(ByVal offset As Integer) As Boolean
```

C# syntax

```
public bool MoveRelative(int offset)
```


Parameters

- **offset** The number of rows to move. Negative values correspond to moving backward.

Returns

True if successful, false otherwise. For example, the method fails if it positions beyond the first or last row.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

If the row does not exist, the method returns false, and the cursor position is after the last row (ULDataReader.IsEOF) if *offset* is positive, and before the first row (ULDataReader.IsBOF) if the *offset* is negative.

See also

- "IsEOF property" on page 254
- "IsBOF property" on page 253

NextResult method

Advances the ULDataReader to the next result when reading the results of batch SQL statements.

Visual Basic syntax

```
Public Overrides Function NextResult() As Boolean
```

C# syntax

```
public override bool NextResult()
```

Returns

True if there are more result sets, false otherwise. For UltraLite.NET, always returns false.

Exceptions

- **"ULException class"** The ULDataReader is not opened.

Remarks

UL Ext: UltraLite.NET does not support batches of SQL statements, hence the ULDataReader is always positioned on the first and only result set. Calling NextResult has no effect.

Read method

Positions the cursor to the next row, or after the last row if the cursor was already on the last row.

Visual Basic syntax

```
Public Overrides Function Read() As Boolean
```

C# syntax

```
public override bool Read()
```

Returns

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

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

This method is identical to the ULDataReader.MoveNext method.

See also

- "MoveNext method" on page 249

Depth property

Returns the depth of nesting for the current row.

Visual Basic syntax

```
Public ReadOnly Overrides Property Depth As Integer
```

C# syntax

```
public override int Depth {get;}
```

Exceptions

- **"ULException class"** The ULDataReader is not opened.

Remarks

The outermost table has a depth of zero.

All UltraLite.NET result sets have a depth of zero.

FieldCount property

Returns the number of columns in the cursor.

Visual Basic syntax

```
Public ReadOnly Overrides Property FieldCount As Integer
```


C# syntax

```
public override int FieldCount {get;}
```

Returns

The number of columns in the cursor as an integer. Returns 0 if the cursor is closed.

Remarks

This method is identical to the ULCursorSchema.ColumnCount method.

See also

- "ColumnCount property" on page 195

HasRows property

Checks whether the ULDataReader has one or more rows.

Visual Basic syntax

```
Public ReadOnly Overrides Property HasRows As Boolean
```

C# syntax

```
public override bool HasRows {get;}
```

Remarks

True if the result set has at least one row, false if there are no rows.

IsBOF property

UL Ext: Checks whether the current row position is before the first row.

Visual Basic syntax

```
Public ReadOnly Property IsBOF As Boolean
```

C# syntax

```
public bool IsBOF {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

True if the current row position is before the first row, false otherwise.

IsClosed property

Checks whether the cursor is currently open.

Visual Basic syntax

```
Public ReadOnly Overrides Property IsClosed As Boolean
```

C# syntax

```
public override bool IsClosed {get;}
```

Remarks

True if the cursor is currently open, false if the cursor is closed.

IsEOF property

UL Ext: Checks whether the current row position is after the last row.

Visual Basic syntax

```
Public ReadOnly Property IsEOF As Boolean
```

C# syntax

```
public bool IsEOF {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

True if the current row position is after the last row, false otherwise.

RecordsAffected property

Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.

Visual Basic syntax

```
Public ReadOnly Overrides Property RecordsAffected As Integer
```

C# syntax

```
public override int RecordsAffected {get;}
```

Remarks

For SELECT statements or CommandType.TableDirect tables, this value is -1.

The number of rows changed, inserted, or deleted by execution of the SQL statement.

See also

- System.Data.CommandType

RowCount property

UL Ext: Returns the number of rows in the cursor.

Visual Basic syntax

```
Public ReadOnly Property RowCount As Integer
```

C# syntax

```
public int RowCount {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The number of rows in the cursor.

One use for RowCount is to decide when to delete old rows to save space. Old rows can be deleted from the UltraLite database without being deleted from the consolidated database using the ULConnection.StopSynchronizationDelete method.

See also

- "StartSynchronizationDelete method" on page 140
- "StopSynchronizationDelete method" on page 141

Schema property

UL Ext: Holds the schema of this cursor.

Visual Basic syntax

```
Public ReadOnly Property Schema As ULCursorSchema
```

C# syntax

```
public ULCursorSchema Schema {get;}
```

Remarks

For result sets, the ULResultSetSchema object representing the schema of the result set. For tables, the ULTableSchema object representing the schema of the table.

This property represents the complete schema of the cursor, including UltraLite.NET extended information which is not represented in the results from ULDataReader.GetSchemaTable.

See also

- "ULTableSchema class" on page 417
- "GetSchemaTable method" on page 241
- "ULResultSetSchema class" on page 358

this property

Returns the value of the specified column in its native format.

Overload list

Name	Description
"this[int] property"	Returns the value of the specified column in its native format.
"this[string] property"	Returns the value of the specified named column in its native format.

this[int] property

Returns the value of the specified column in its native format.

Visual Basic syntax

```
Public ReadOnly Overrides Property Item(
    ByVal colID As Integer
) As Object
```

C# syntax

```
public override object this[int colID] {get;}
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns

The column value as the .NET type most appropriate for the column or DBNull if column is NULL.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

In C#, this property is the indexer for the ULDataReader class.

This method is identical in functionality to the ULDataReader.GetValue(int) method.

See also

- "GetFieldType method" on page 235
- "GetValue method" on page 246
- "FieldCount property" on page 252

this[string] property

Returns the value of the specified named column in its native format.

Visual Basic syntax

```
Public ReadOnly Overrides Property Item(ByVal name As String) As Object
```

C# syntax

```
public override object this[string name] {get;}
```

Parameters

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

Returns

The column value as the .NET type most appropriate for the column or DBNull if column is NULL.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

In C#, this property is the indexer for the ULDataReader class.

Note that in result sets, not all columns have names and not all column names are unique. If you are not using aliases, the name of a non-computed column is prefixed with the name of the table the column is from. For example, MyTable.ID is the name of the only column in the result set for the query "SELECT ID FROM MyTable".

When accessing columns multiple times, it is more efficient to access columns by column ID than by name.

This method is equivalent to:

```
dataReader.GetValue( dataReader.GetOrdinal( name ) )
```

See also

- "GetOrdinal method" on page 239
- "GetValue method" on page 246
- "GetFieldType method" on page 235

ULException class

Represents a SQL error returned by the UltraLite.NET database.

Visual Basic syntax

```
Public NotInheritable Class ULException
    Inherits System.ApplicationException
```

C# syntax

```
public sealed class ULException : System.ApplicationException
```

Base classes

- System.ApplicationException

Members

All members of ULException class, including all inherited members.

Name	Description
"GetObjectData method"	Populates a SerializationInfo with the data needed to serialize this ULException.
"NativeError property"	Returns the SQLCODE returned by the database.
"Source property"	Returns the name of the provider that generated the error.

Remarks

The SQLCODE denoting the error is returned in the NativeError.

This class is not serializable under the .NET Compact Framework.

See also

- "NativeError property" on page 259

GetObjectData method

Populates a SerializationInfo with the data needed to serialize this ULException.

Visual Basic syntax

```
Public Overrides Sub GetObjectData(
    ByVal info As SerializationInfo,
    ByVal context As StreamingContext
)
```

C# syntax

```
public override void GetObjectData(
    SerializationInfo info,
```



```
    StreamingContext context
)
```

Parameters

- **info** The SerializationInfo to populate with data.
- **context** The destination for this serialization.

Remarks

This method is not supported under the .NET Compact Framework.

NativeError property

Returns the SQLCODE returned by the database.

Visual Basic syntax

```
Public ReadOnly Property NativeError As ULSQLCode
```

C# syntax

```
public ULSQLCode NativeError {get;}
```

Remarks

The ULSQLCode value returned by the database.

Source property

Returns the name of the provider that generated the error.

Visual Basic syntax

```
Public ReadOnly Overrides Property Source As String
```

C# syntax

```
public override string Source {get;}
```

Remarks

The string value identifying UltraLite.NET as the provider.

ULFactory class

Represents a set of methods for creating instances of the iAnywhere.Data.UltraLite provider's implementation of the data source classes.

Visual Basic syntax

```
Public NotInheritable Class ULFactory
    Inherits System.Data.Common.DbProviderFactory
```

C# syntax

```
public sealed class ULFactory : System.Data.Common.DbProviderFactory
```

Base classes

- System.Data.Common.DbProviderFactory

Members

All members of ULFactory class, including all inherited members.

Name	Description
"CreateCommand method"	Returns a strongly typed System.Data.Common.DbCommand instance.
"CreateCommandBuilder method"	Returns a strongly typed System.Data.Common.DbCommandBuilder instance.
"CreateConnection method"	Returns a strongly typed System.Data.Common.DbConnection instance.
"CreateConnectionStringBuilder method"	Returns a strongly typed System.Data.Common.DbConnectionStringBuilder instance.
"CreateDataAdapter method"	Returns a strongly typed System.Data.Common.DbDataAdapter instance.
CreateDataSourceEnumerator method (Inherited from System.Data.Common.DbProviderFactory)	Returns a new instance of the provider's class that implements the System.Data.Common.DbDataSourceEnumerator class.
"CreateParameter method"	Returns a strongly typed System.Data.Common.DbParameter instance.
CreatePermission method (Inherited from System.Data.Common.DbProviderFactory)	Returns a new instance of the provider's class that implements the provider's version of the System.Security.CodeAccessPermission class.
"CanCreateDataSourceEnumerator property"	Returns false to indicate the UltraLite.NET does not support DbDataSourceEnumerator.
"Instance field"	Represents the singleton instance of the ULFactory class.

Remarks

The ULFactory class is not available in the .NET Compact Framework 2.0.

ADO.NET 2.0 adds two new classes, the System.Data.Common.DbProviderFactories class and the System.Data.Common.DbProviderFactory class, to make provider independent code easier to write. To use them with UltraLite.NET specify iAnywhere.Data.UltraLite as the provider invariant name passed to GetFactory. For example:

```
' Visual Basic
Dim factory As DbProviderFactory = _
    DbProviderFactories.GetFactory( "iAnywhere.Data.UltraLite" )
Dim conn As DbConnection = _
    factory.CreateConnection()
```

The following code is the C# language equivalent:

```
// C#
DbProviderFactory factory =
    DbProviderFactories.GetFactory( "iAnywhere.Data.UltraLite" );
DbConnection conn = factory.CreateConnection();
```

In this example, conn is created as an ULConnection object.

For an explanation of provider factories and generic programming in ADO.NET 2.0, see http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/vsgenerics.asp.

UltraLite.NET does not support CreateCommandBuilder(), CreateDataSourceEnumerator(), and CreatePermission().

See also

- System.Data.Common.DbProviderFactories

CreateCommand method

Returns a strongly typed System.Data.Common.DbCommand instance.

Visual Basic syntax

```
Public Overrides Function CreateCommand() As DbCommand
```

C# syntax

```
public override DbCommand CreateCommand()
```

Returns

A new ULCommand instance typed as DbCommand.

See also

- “ULCommand class” on page 64
- System.Data.Common.DbCommand

CreateCommandBuilder method

Returns a strongly typed System.Data.Common.DbCommandBuilder instance.

Visual Basic syntax

```
Public Overrides Function CreateCommandBuilder() As DbCommandBuilder
```

C# syntax

```
public override DbCommandBuilder CreateCommandBuilder()
```

Returns

A new ULCommandBuilder instance typed as DbCommandBuilder.

See also

- "ULCommandBuilder class" on page 99
- System.Data.Common.DbCommandBuilder

CreateConnection method

Returns a strongly typed System.Data.Common.DbConnection instance.

Visual Basic syntax

```
Public Overrides Function CreateConnection() As DbConnection
```

C# syntax

```
public override DbConnection CreateConnection()
```

Returns

A new ULConnection instance typed as DbConnection.

See also

- "ULConnection class" on page 110
- System.Data.Common.DbConnection

CreateConnectionStringBuilder method

Returns a strongly typed System.Data.Common.DbConnectionStringBuilder instance.

Visual Basic syntax

```
Public Overrides Function CreateConnectionStringBuilder()
    As DbConnectionStringBuilder
```


C# syntax

```
public override DbConnectionStringBuilder CreateConnectionStringBuilder()
```

Returns

A new ULConnectionStringBuilder instance typed as DbConnectionStringBuilder.

See also

- "ULConnectionStringBuilder class" on page 164
- System.Data.Common.DbConnectionStringBuilder

CreateDataAdapter method

Returns a strongly typed System.Data.Common.DbDataAdapter instance.

Visual Basic syntax

```
Public Overrides Function CreateDataAdapter() As DbDataAdapter
```

C# syntax

```
public override DbDataAdapter CreateDataAdapter()
```

Returns

A new ULDataAdapter instance typed as DbDataAdapter.

See also

- "ULDataAdapter class" on page 196
- System.Data.Common.DbDataAdapter

CreateParameter method

Returns a strongly typed System.Data.Common.DbParameter instance.

Visual Basic syntax

```
Public Overrides Function CreateParameter() As DbParameter
```

C# syntax

```
public override DbParameter CreateParameter()
```

Returns

A new ULParameter instance typed as DbParameter.

See also

- "ULParameter class" on page 301
- System.Data.Common.DbParameter

CanCreateDataSourceEnumerator property

Returns false to indicate the UltraLite.NET does not support DbDataSourceEnumerator.

Visual Basic syntax

```
Public ReadOnly Overrides Property CanCreateDataSourceEnumerator As
Boolean
```

C# syntax

```
public override bool CanCreateDataSourceEnumerator {get;}
```

Remarks

False to indicate that ULFactory does not implement the CreateDataSourceEnumerator() method.

Instance field

Represents the singleton instance of the ULFactory class.

Visual Basic syntax

```
Public Shared ReadOnly Instance As ULFactory
```

C# syntax

```
public static readonly ULFactory Instance;
```

Remarks

The ULFactory class is not available in the .NET Compact Framework 2.0.

ULFactory is a singleton class, which means only this instance of this class can exist.

Normally you would not use this field directly. Instead, you get a reference to this instance of ULFactory using System.Data.Common.DbProviderFactories.GetFactory(String).

See also

- "ULFactory class" on page 259

ULFileTransfer class

UL Ext: Transfers a file from a remote database using the MobiLink server.

Visual Basic syntax

```
Public NotInheritable Class ULFileTransfer
```

C# syntax

```
public sealed class ULFileTransfer
```

Members

All members of ULFileTransfer class, including all inherited members.

Name	Description
"ULFileTransfer constructor"	Initializes a ULFileTransfer object.
"DownloadFile method"	Download the file specified by the properties of this object.
"UploadFile method"	Upload the file specified by the properties of this object.
"AuthenticationParms property"	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
"AuthStatus property"	Returns the authorization status code for the last file transfer attempt.
"AuthValue property"	Returns the return value from custom user authentication synchronization scripts.
"FileAuthCode property"	Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.
"FileName property"	Specifies the name of the file to download.
"LocalFileName property"	Specifies the local file name for the downloaded file.
"LocalPath property"	Specifies where to download the file.
"Password property"	The MobiLink password for the user specified by UserName.
"RemoteKey property"	The key that uniquely identifies the MobiLink client to the MobiLink server.
"ResumePartialDownload property"	Specifies whether to resume or discard a previous partial download.
"Stream property"	Specifies the MobiLink synchronization stream to use for the file transfer.
"StreamErrorCode property"	Returns the error reported by the stream itself for the last file transfer attempt.

Name	Description
"StreamErrorSystem property"	Returns the stream error system-specific code.
"StreamParms property"	Specifies the parameters to configure the synchronization stream.
"TransferedFile property"	Checks whether the file was actually downloaded during the last file transfer attempt.
"UserName property"	The user name that identifies the MobiLink client to the MobiLink server.
"Version property"	Specifies which synchronization script to use.

Remarks

You do not need a database connection to perform a file transfer, however, if your application uses an UltraLite database with the UltraLite Engine runtime, you must set ULDatabaseManager.RuntimeType to the appropriate value before using this API or any other UltraLite.NET API.

To transfer a file you must set the ULFileTransfer.FileName, ULFileTransfer.Stream, ULFileTransfer.UserName, and ULFileTransfer.Version.

See also

- "FileName property" on page 274
- "Stream property" on page 277
- "UserName property" on page 279
- "Version property" on page 279

ULFileTransfer constructor

Initializes a ULFileTransfer object.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ULFileTransfer()
```

Remarks

The connection must be opened before you can perform any operations against the database.

You do not need a database connection to perform a file transfer, however, if your application uses an UltraLite database with the UltraLite Engine runtime, you must set ULDatabaseManager.RuntimeType to the appropriate value before using this API or any other UltraLite.NET API.

The ULFileTransfer object needs to have the ULFileTransfer.FileName, ULFileTransfer.Stream, ULFileTransfer.UserName, and ULFileTransfer.Version set before it can transfer a file.

See also

- "DownloadFile method" on page 267
- "UploadFile method" on page 269
- "FileName property" on page 274
- "Stream property" on page 277
- "UserName property" on page 279
- "Version property" on page 279

DownloadFile method

Download the file specified by the properties of this object.

Overload list

Name	Description
"DownloadFile() method"	Download the file specified by the properties of this object.
"DownloadFile(ULFileTransfer-ProgressListener) method"	Download the file specified by the properties of this object with progress events posted to the specified listener.

DownloadFile() method

Download the file specified by the properties of this object.

Visual Basic syntax

```
Public Function DownloadFile() As Boolean
```

C# syntax

```
public bool DownloadFile()
```

Returns

True if successful, false otherwise (check ULFileTransfer.StreamErrorCode and other status properties for reason).

Remarks

The file specified by the ULFileTransfer.FileName is downloaded by the MobiLink server to the ULFileTransfer.LocalPath using the ULFileTransfer.Stream, ULFileTransfer.UserName, ULFileTransfer.Password, and ULFileTransfer.Version. Other properties that affect the download are ULFileTransfer.LocalFileName, ULFileTransfer.AuthenticationParms, and ULFileTransfer.ResumePartialDownload.

To avoid file corruption, UltraLite.NET downloads to a temporary file and only replaces the local file once the download has completed.

A detailed result status is reported in this object's ULFileTransfer.AuthStatus, ULFileTransfer.AuthValue, ULFileTransfer.FileAuthCode, ULFileTransfer.TransferedFile, ULFileTransfer.StreamErrorCode, and ULFileTransfer.StreamErrorSystem.

See also

- "DownloadFile method" on page 267
- "FileName property" on page 274
- "LocalPath property" on page 275
- "Stream property" on page 277
- "UserName property" on page 279
- "Password property" on page 275
- "Version property" on page 279
- "LocalFileName property" on page 274
- "AuthenticationParms property" on page 272
- "ResumePartialDownload property" on page 276
- "AuthStatus property" on page 273
- "AuthValue property" on page 273
- "FileAuthCode property" on page 273
- "TransferedFile property" on page 278
- "StreamErrorCode property" on page 277
- "StreamErrorSystem property" on page 278
- "StreamErrorCode property" on page 277
- "UploadFile method" on page 269

DownloadFile(ULFileTransferProgressListener) method

Download the file specified by the properties of this object with progress events posted to the specified listener.

Visual Basic syntax

```
Public Function DownloadFile(
    ByVal listener As ULFileTransferProgressListener
) As Boolean
```

C# syntax

```
public bool DownloadFile(ULFileTransferProgressListener listener)
```

Parameters

- **listener** The object that receives file transfer progress events.

Returns

True if successful, false otherwise (check ULFileTransfer.StreamErrorCode and other status properties for reason).

Remarks

The file specified by the ULFileTransfer.FileName is downloaded by the MobiLink server to the ULFileTransfer.LocalPath using the ULFileTransfer.Stream, ULFileTransfer.UserName, ULFileTransfer.Password, and ULFileTransfer.Version. Other properties that affect the download are ULFileTransfer.LocalFileName, ULFileTransfer.AuthenticationParms, and ULFileTransfer.ResumePartialDownload.

To avoid file corruption, UltraLite.NET downloads to a temporary file and only replaces the local file once the download has completed.

A detailed result status is reported in this object's ULFileTransfer.AuthStatus, ULFileTransfer.AuthValue, ULFileTransfer.FileAuthCode, ULFileTransfer.TransferedFile, ULFileTransfer.StreamErrorCode, and ULFileTransfer.StreamErrorSystem.

Errors may result in no data being sent to the listener.

See also

- “FileName property” on page 274
- “LocalPath property” on page 275
- “Stream property” on page 277
- “UserName property” on page 279
- “Password property” on page 275
- “Version property” on page 279
- “LocalFileName property” on page 274
- “AuthenticationParms property” on page 272
- “ResumePartialDownload property” on page 276
- “AuthStatus property” on page 273
- “AuthValue property” on page 273
- “FileAuthCode property” on page 273
- “TransferedFile property” on page 278
- “StreamErrorCode property” on page 277
- “StreamErrorSystem property” on page 278
- “StreamErrorCode property” on page 277
- “UploadFile method” on page 269

UploadFile method

Upload the file specified by the properties of this object.

Overload list

Name	Description
“UploadFile() method”	Upload the file specified by the properties of this object.

Name	Description
"UploadFile(ULFileTransfer-ProgressListener) method"	Upload the file specified by the properties of this object with progress events posted to the specified listener.

UploadFile() method

Upload the file specified by the properties of this object.

Visual Basic syntax

```
Public Function UploadFile() As Boolean
```

C# syntax

```
public bool UploadFile()
```

Returns

True if successful, false otherwise (check ULFileTransfer.StreamErrorCode and other status properties for reason).

Remarks

The file specified by the ULFileTransfer.FileName is uploaded to the MobiLink server from the ULFileTransfer.LocalPath using the ULFileTransfer.Stream, ULFileTransfer.UserName, ULFileTransfer.Password, and ULFileTransfer.Version. Other properties that affect the download are ULFileTransfer.LocalFileName, ULFileTransfer.AuthenticationParms, and ULFileTransfer.ResumePartialDownload.

A detailed result status is reported in this object's ULFileTransfer.AuthStatus, ULFileTransfer.AuthValue, ULFileTransfer.FileAuthCode, ULFileTransfer.TransferedFile, ULFileTransfer.StreamErrorCode, and ULFileTransfer.StreamErrorSystem.

See also

- "UploadFile method" on page 269
- "FileName property" on page 274
- "LocalPath property" on page 275
- "Stream property" on page 277
- "UserName property" on page 279
- "Password property" on page 275
- "Version property" on page 279
- "LocalFileName property" on page 274
- "AuthenticationParms property" on page 272
- "ResumePartialDownload property" on page 276
- "AuthStatus property" on page 273
- "AuthValue property" on page 273
- "FileAuthCode property" on page 273
- "TransferedFile property" on page 278
- "StreamErrorCode property" on page 277
- "StreamErrorSystem property" on page 278
- "StreamErrorCode property" on page 277

UploadFile(ULFileTransferProgressListener) method

Upload the file specified by the properties of this object with progress events posted to the specified listener.

Visual Basic syntax

```
Public Function UploadFile(
    ByVal listener As ULFileTransferProgressListener
) As Boolean
```

C# syntax

```
public bool UploadFile(ULFileTransferProgressListener listener)
```

Parameters

- **listener** The object that receives file transfer progress events.

Returns

True if successful, false otherwise (check ULFileTransfer.StreamErrorCode and other status properties for reason).

Remarks

The file specified by the ULFileTransfer.FileName is uploaded to the MobiLink server from the ULFileTransfer.LocalPath using the ULFileTransfer.Stream, ULFileTransfer.UserName, ULFileTransfer.Password, and ULFileTransfer.Version. Other properties that affect the download are ULFileTransfer.LocalFileName, ULFileTransfer.AuthenticationParms, and ULFileTransfer.ResumePartialDownload.

A detailed result status is reported in this object's ULFileTransfer.AuthStatus, ULFileTransfer.AuthValue, ULFileTransfer.FileAuthCode, ULFileTransfer.TransferedFile, ULFileTransfer.StreamErrorCode, and ULFileTransfer.StreamErrorSystem.

Errors may result in no data being sent to the listener.

See also

- "UploadFile method" on page 269
- "FileName property" on page 274
- "LocalPath property" on page 275
- "Stream property" on page 277
- "UserName property" on page 279
- "Password property" on page 275
- "Version property" on page 279
- "LocalFileName property" on page 274
- "AuthenticationParms property" on page 272
- "ResumePartialDownload property" on page 276
- "AuthStatus property" on page 273
- "AuthValue property" on page 273
- "FileAuthCode property" on page 273
- "TransferedFile property" on page 278
- "StreamErrorCode property" on page 277
- "StreamErrorSystem property" on page 278
- "StreamErrorCode property" on page 277

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Visual Basic syntax

```
Public Property AuthenticationParms As String()
```

C# syntax

```
public String[] AuthenticationParms {get;set;}
```

Remarks

An array of strings, each containing an authentication parameter (null array entries result in a synchronization error). The default is a null reference (Nothing in Visual Basic), meaning no authentication parameters.

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).

AuthStatus property

Returns the authorization status code for the last file transfer attempt.

Visual Basic syntax

```
Public ReadOnly Property AuthStatus As ULAuthStatusCode
```

C# syntax

```
public ULAuthStatusCode AuthStatus {get;}
```

Remarks

One of the ULAuthStatusCode values denoting the authorization status for the last file transfer attempt.

See also

- "ULAuthStatusCode enumeration" on page 435

AuthValue property

Returns the return value from custom user authentication synchronization scripts.

Visual Basic syntax

```
Public ReadOnly Property AuthValue As Long
```

C# syntax

```
public long AuthValue {get;}
```

Remarks

A long integer returned from custom user authentication synchronization scripts.

FileAuthCode property

Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.

Visual Basic syntax

```
Public ReadOnly Property FileAuthCode As UShort
```

C# syntax

```
public ushort FileAuthCode {get;}
```

Remarks

An unsigned short integer returned from the authenticate_file_transfer script for the last file transfer attempt.

FileName property

Specifies the name of the file to download.

Visual Basic syntax

```
Public Property FileName As String
```

C# syntax

```
public string FileName {get;set;}
```

Remarks

A string specifying the name of the file as recognized by the MobiLink server. This property has no default value, and must be explicitly set.

FileName is the name of the file on the server running MobiLink. MobiLink first searches for this file in the UserName subdirectory and then in the root directory (the root directory is specified via the MobiLink server's -ftr option). FileName must not include any drive or path information so that the MobiLink server can find it. For example, "myfile.txt" is valid, but "somedir\myfile.txt", "..\myfile.txt", and "c:\myfile.txt" are all invalid.

See also

- “LocalFileName property” on page 274
- “DownloadFile method” on page 267
- “UploadFile method” on page 269

LocalFileName property

Specifies the local file name for the downloaded file.

Visual Basic syntax

```
Public Property LocalFileName As String
```

C# syntax

```
public string LocalFileName {get;set;}
```

Remarks

A string specifying the local file name for the downloaded file. If the value is a null reference (Nothing in Visual Basic), FileName is used. The default is a null reference (Nothing in Visual Basic).

See also

- “FileName property” on page 274

LocalPath property

Specifies where to download the file.

Visual Basic syntax

```
Public Property LocalPath As String
```

C# syntax

```
public string LocalPath {get;set;}
```

Remarks

A string specifying the local directory of the file. The default is a null reference (Nothing in Visual Basic).

The default local directory varies depending on the device's operating system:

- For Windows Mobile devices, if the LocalPath is a null reference (Nothing in Visual Basic), the file is stored in the root (\) directory.
- For desktop applications, if the LocalPath is a null reference (Nothing in Visual Basic), the file is stored in the current directory.

See also

- "DownloadFile method" on page 267
- "UploadFile method" on page 269

Password property

The MobiLink password for the user specified by UserName.

Visual Basic syntax

```
Public Property Password As String
```

C# syntax

```
public string Password {get;set;}
```

Remarks

A string specifying the MobiLink password. The default is a null reference (Nothing in Visual Basic), meaning no password is specified.

The MobiLink user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink server.

See also

- "UserName property" on page 279
- "DownloadFile method" on page 267
- "UploadFile method" on page 269

RemoteKey property

The key that uniquely identifies the MobiLink client to the MobiLink server.

Visual Basic syntax

```
Public Property RemoteKey As String
```

C# syntax

```
public string RemoteKey {get;set;}
```

Remarks

A string specifying the remote key. This property has no default value, and must be explicitly set.

The MobiLink server passes this value to various scripts to uniquely identify this client.

See also

- "DownloadFile method" on page 267
- "UploadFile method" on page 269

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Visual Basic syntax

```
Public Property ResumePartialDownload As Boolean
```

C# syntax

```
public bool ResumePartialDownload {get;set;}
```

Remarks

True to resume a previous partial download, false to discard a previous partial download. The default is false.

UltraLite.NET has the ability to restart downloads that fail because of communication errors or user aborts through the ULFileTransferListener. UltraLite.NET processes the download as it is received. If a download is interrupted, then the partially download file is retained and can be resumed during the next file transfer.

If the file has been updated on the server, a partial download is discarded and a new download starts.

See also

- "DownloadFile method" on page 267
- "UploadFile method" on page 269

Stream property

Specifies the MobiLink synchronization stream to use for the file transfer.

Visual Basic syntax

```
Public Property Stream As ULStreamType
```

C# syntax

```
public ULStreamType Stream {get;set;}
```

Remarks

One of the ULStreamType values specifying the type of synchronization stream to use. The default is ULStreamType.TCPIP.

Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. These parameters are supplied by the ULFileTransfer.StreamParms.

If the stream type is set to a value that is invalid for the platform, the stream type is set to ULStreamType.TCPIP.

See also

- "ULStreamType enumeration" on page 443
- "StreamParms property" on page 278
- "DownloadFile method" on page 267
- "UploadFile method" on page 269

StreamErrorCode property

Returns the error reported by the stream itself for the last file transfer attempt.

Visual Basic syntax

```
Public ReadOnly Property StreamErrorCode As ULStreamErrorCode
```

C# syntax

```
public ULStreamErrorCode StreamErrorCode {get;}
```

Remarks

One of the ULStreamErrorCode values denoting the error reported by the stream itself, ULStreamErrorCode.NONE if no error occurred.

StreamErrorSystem property

Returns the stream error system-specific code.

Visual Basic syntax

```
Public ReadOnly Property StreamErrorSystem As Integer
```

C# syntax

```
public int StreamErrorSystem {get;}
```

Remarks

An integer denoting the stream error system-specific code.

StreamParms property

Specifies the parameters to configure the synchronization stream.

Visual Basic syntax

```
Public Property StreamParms As String
```

C# syntax

```
public string StreamParms {get;set;}
```

Remarks

A string, in the form of a semicolon-separated list of keyword-value pairs, specifying the parameters for the stream. The default is a null reference. (Nothing in Visual Basic)

For information about configuring specific stream types, see "Network protocol options for UltraLite synchronization streams" [*UltraLite - Database Management and Reference*].

StreamParms is 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").

See also

- "Stream property" on page 277
- "DownloadFile method" on page 267
- "UploadFile method" on page 269
- "ULStreamType enumeration" on page 443

TransferedFile property

Checks whether the file was actually downloaded during the last file transfer attempt.

Visual Basic syntax

```
Public ReadOnly Property TransferedFile As Boolean
```

C# syntax

```
public bool TransferedFile {get;}
```

Remarks

True if the file was downloaded, false otherwise.

If the file is already up-to-date when the DownloadFile() or UploadFile() is invoked, it returns true, but TransferedFile is false. If an error occurs and DownloadFile() or UploadFile() returns false, TransferedFile is false.

See also

- "DownloadFile method" on page 267
- "UploadFile method" on page 269

UserName property

The user name that identifies the MobiLink client to the MobiLink server.

Visual Basic syntax

```
Public Property UserName As String
```

C# syntax

```
public string UserName {get;set;}
```

Remarks

A string specifying the user name. This property has no default value, and must be explicitly set.

The MobiLink server uses this value to locate the file to download. The MobiLink user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink server.

See also

- "Password property" on page 275
- "DownloadFile method" on page 267
- "UploadFile method" on page 269

Version property

Specifies which synchronization script to use.

Visual Basic syntax

```
Public Property Version As String
```

C# syntax

```
public string Version {get;set;}
```

Remarks

A string specifying the version of the synchronization script to use. This property has no default value, and must be explicitly set.

Each synchronization script in the consolidated database is marked with a version string. The version string allows an UltraLite application to choose from a set of synchronization scripts.

See also

- "DownloadFile method" on page 267
- "UploadFile method" on page 269

ULFileTransferProgressData class

UL Ext: Returns file transfer progress monitoring data.

Visual Basic syntax

```
Public Class ULFileTransferProgressData
```

C# syntax

```
public class ULFileTransferProgressData
```

Members

All members of ULFileTransferProgressData class, including all inherited members.

Name	Description
"BytesReceived property"	Returns the number of bytes received so far.
"FileSize property"	Returns the size of the file being transferred.
"Flags property"	Returns the current file transfer flags indicating additional information relating to the current state.
"ResumedAtSize property"	Returns the point in the file where the transfer was resumed.
"FLAG_IS_BLOCKING field"	A flag indicating that the file transfer is blocked awaiting a response from the MobiLink server.

See also

- "ULFileTransferProgressListener interface" on page 282

BytesReceived property

Returns the number of bytes received so far.

Visual Basic syntax

```
Public ReadOnly Property BytesReceived As ULong
```

C# syntax

```
public ulong BytesReceived {get;}
```

Remarks

The number of bytes received so far.

FileSize property

Returns the size of the file being transferred.

Visual Basic syntax

```
Public ReadOnly Property FileSize As ULong
```

C# syntax

```
public ulong FileSize {get;}
```

Remarks

The size of the file in bytes.

Flags property

Returns the current file transfer flags indicating additional information relating to the current state.

Visual Basic syntax

```
Public ReadOnly Property Flags As Integer
```

C# syntax

```
public int Flags {get;}
```

Remarks

An integer containing a combination of flags or'ed together.

See also

- "FLAG_IS_BLOCKING field" on page 282

ResumedAtSize property

Returns the point in the file where the transfer was resumed.

Visual Basic syntax

```
Public ReadOnly Property ResumedAtSize As ULong
```

C# syntax

```
public ulong ResumedAtSize {get;}
```

Remarks

The number of bytes transferred previously.

FLAG_IS_BLOCKING field

A flag indicating that the file transfer is blocked awaiting a response from the MobiLink server.

Visual Basic syntax

```
Public Const FLAG_IS_BLOCKING As Integer
```

C# syntax

```
public const int FLAG_IS_BLOCKING;
```

ULFileTransferProgressListener interface

UL Ext: The listener interface for receiving file transfer progress events.

Visual Basic syntax

```
Public Interface ULFileTransferProgressListener
```

C# syntax

```
public interface ULFileTransferProgressListener
```

Members

All members of ULFileTransferProgressListener interface, including all inherited members.

Name	Description
"FileTransferProgressed method"	Invoked during a file transfer to inform the user of progress.

See also

- "DownloadFile method" on page 267

FileTransferProgressed method

Invoked during a file transfer to inform the user of progress.

Visual Basic syntax

```
Public Function FileTransferProgressed(
    ByVal data As ULFileTransferProgressData
) As Boolean
```

C# syntax

```
public bool FileTransferProgressed(ULFileTransferProgressData data)
```

Parameters

- **data** A ULFileTransferProgressData object containing the latest file transfer progress data.

Returns

This method should return true to cancel the transfer or return false to continue.

Remarks

This method should return true to cancel the transfer or return false to continue.

No UltraLite.NET API methods should be invoked during a FileTransferProgressed call.

See also

- "ULFileTransferProgressData class" on page 280

ULIndexSchema class

UL Ext: Represents the schema of an UltraLite table index.

Visual Basic syntax

```
Public NotInheritable Class ULIndexSchema
```

C# syntax

```
public sealed class ULIndexSchema
```

Members

All members of ULIndexSchema class, including all inherited members.

Name	Description
"GetColumnName method"	Returns the name of the *colOrdinalInIndex* 'th column in this index.
"IsColumnDescending method"	Checks whether the named column is used in descending order by the index.
"ColumnCount property"	Returns the number of columns in the index.
"IsForeignKey property"	Checks whether the index is a foreign key.
"IsForeignKeyCheckOnCommit property"	Checks whether referential integrity for the foreign key is performed on commits or on inserts and updates.
"IsForeignKeyNullable property"	Checks whether the foreign key is nullable.
"IsOpen property"	Determines whether the index schema is open or closed.
"IsPrimaryKey property"	Checks whether the index is the primary key.
"IsUniqueIndex property"	Checks whether the index is unique.
"IsUniqueKey property"	Checks whether the index is a unique key.
"Name property"	Returns the name of the index.
"ReferencedIndexName property"	The name of the referenced primary index if the index is a foreign key.
"ReferencedTableName property"	The name of the referenced primary table if the index is a foreign key.

Remarks

There is no constructor for this class. Index schemas are created using the ULTableSchema.PrimaryKey, ULTableSchema.GetIndex(string), and ULTableSchema.GetOptimalIndex(int) of the ULTableSchema.

See also

- "PrimaryKey property" on page 429
- "GetIndex method" on page 420
- "GetOptimalIndex method" on page 421
- "ULTableSchema class" on page 417

GetColumnName method

Returns the name of the *colOrdinalInIndex* 'th column in this index.

Visual Basic syntax

```
Public Function GetColumnName(
    ByVal colOrdinalInIndex As Short
) As String
```

C# syntax

```
public string GetColumnName(short colOrdinalInIndex)
```

Parameters

- **colOrdinalInIndex** The ordinal of the desired column in the index. The value must be in the range [1,ColumnCount].

Returns

The name of the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Column ordinals and count may change during a schema upgrade. Column ordinals from an index are different than the column IDs in a table or another index, even if they refer to the same physical column in a particular table.

See also

- "ColumnCount property" on page 286

IsColumnDescending method

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

Visual Basic syntax

```
Public Function IsColumnDescending(ByVal name As String) As Boolean
```

C# syntax

```
public bool IsColumnDescending(string name)
```

Parameters

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

Returns

True if the column is used in descending order, false if the column is used in ascending order.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetColumnName method" on page 285
- "ColumnCount property" on page 286

ColumnCount property

Returns the number of columns in the index.

Visual Basic syntax

```
Public ReadOnly Property ColumnCount As Short
```

C# syntax

```
public short ColumnCount {get;}
```

Remarks

The number of columns in the index.

Column ordinals in indexes range from 1 to ColumnCount, inclusive.

Column ordinals and count may change during a schema upgrade. Column ordinals from an index are different than the column IDs in a table or another index, even if they refer to the same physical column in a particular table.

IsForeignKey property

Checks whether the index is a foreign key.

Visual Basic syntax

```
Public ReadOnly Property IsForeignKey As Boolean
```

C# syntax

```
public bool IsForeignKey {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

True if the index is the foreign key, false if the index is not the foreign key.

Columns in a foreign key may reference another table's non-null, unique index.

IsForeignKeyCheckOnCommit property

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

Visual Basic syntax

```
Public ReadOnly Property IsForeignKeyCheckOnCommit As Boolean
```

C# syntax

```
public bool IsForeignKeyCheckOnCommit {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred (including index is not a foreign key).

Remarks

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

See also

- "IsForeignKey property" on page 286

IsForeignKeyNullable property

Checks whether the foreign key is nullable.

Visual Basic syntax

```
Public ReadOnly Property IsForeignKeyNullable As Boolean
```

C# syntax

```
public bool IsForeignKeyNullable {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred (including index is not a foreign key).

Remarks

True if the foreign key is nullable, false if the foreign key is not nullable.

See also

- "IsForeignKey property" on page 286

IsOpen property

Determines whether the index schema is open or closed.

Visual Basic syntax

```
Public ReadOnly Property IsOpen As Boolean
```

C# syntax

```
public bool IsOpen {get;}
```

Remarks

True if the index schema is open, otherwise false.

IsPrimaryKey property

Checks whether the index is the primary key.

Visual Basic syntax

```
Public ReadOnly Property IsPrimaryKey As Boolean
```

C# syntax

```
public bool IsPrimaryKey {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

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

Columns in the primary key may not be null.

IsUniqueIndex property

Checks whether the index is unique.

Visual Basic syntax

```
Public ReadOnly Property IsUniqueIndex As Boolean
```

C# syntax

```
public bool IsUniqueIndex {get;}
```


Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

True if the index is unique, false if the index is not unique.

Columns in a unique index may be null.

IsUniqueKey property

Checks whether the index is a unique key.

Visual Basic syntax

```
Public ReadOnly Property IsUniqueKey As Boolean
```

C# syntax

```
public bool IsUniqueKey {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

True if the index is a unique key, false if the index is not a unique key.

Columns in a unique key may not be null.

Name property

Returns the name of the index.

Visual Basic syntax

```
Public ReadOnly Property Name As String
```

C# syntax

```
public string Name {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

A string specifying the name of the index.

ReferencedIndexName property

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

Visual Basic syntax

```
Public ReadOnly Property ReferencedIndexName As String
```

C# syntax

```
public string ReferencedIndexName {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred (including index is not a foreign key).

Remarks

A string specifying the name of the referenced primary index.

See also

- "IsForeignKey property" on page 286

ReferencedTableName property

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

Visual Basic syntax

```
Public ReadOnly Property ReferencedTableName As String
```

C# syntax

```
public string ReferencedTableName {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred (including index is not a foreign key).

Remarks

A string specifying the name of the referenced primary table.

See also

- "IsForeignKey property" on page 286

ULInfoMessageEventArgs class

Provides data for the ULConnection.InfoMessage event.

Visual Basic syntax

```
Public NotInheritable Class ULInfoMessageEventArgs
    Inherits System.EventArgs
```

C# syntax

```
public sealed class ULInfoMessageEventArgs : System.EventArgs
```

Base classes

- System.EventArgs

Members

All members of ULInfoMessageEventArgs class, including all inherited members.

Name	Description
"ToString method"	The string representation of the ULConnection.InfoMessage event.
"Message property"	The informational or warning message string returned by the database.
"NativeError property"	The SQLCODE corresponding to the informational message or warning returned by the database.
"Source property"	The name of the ADO.NET data provider returning the message.
Empty field (Inherited from System.EventArgs)	Represents an event with no event data.

See also

- "InfoMessage event" on page 151

ToString method

The string representation of the ULConnection.InfoMessage event.

Visual Basic syntax

```
Public Overrides Function ToString() As String
```

C# syntax

```
public override string ToString()
```

Returns

The informational or warning message string.

Remarks

A string representation of the ULConnection.InfoMessage event.

See also

- "InfoMessage event" on page 151

Message property

The informational or warning message string returned by the database.

Visual Basic syntax

```
Public ReadOnly Property Message As String
```

C# syntax

```
public string Message {get;}
```

Remarks

A string containing the informational or warning message.

NativeError property

The SQLCODE corresponding to the informational message or warning returned by the database.

Visual Basic syntax

```
Public ReadOnly Property NativeError As ULSQLCode
```

C# syntax

```
public ULSQLCode NativeError {get;}
```

Remarks

An informational or warning ULSQLCode value.

Source property

The name of the ADO.NET data provider returning the message.

Visual Basic syntax

```
Public ReadOnly Property Source As String
```

C# syntax

```
public string Source {get;}
```


Remarks

The string "UltraLite.NET Data Provider".

ULMetaDataCollectionNames class

Provides a list of constants for use with the ULConnection.GetSchema(String,String[]) to retrieve metadata collections.

Visual Basic syntax

```
Public NotInheritable Class ULMetaDataCollectionNames
```

C# syntax

```
public sealed class ULMetaDataCollectionNames
```

Members

All members of ULMetaDataCollectionNames class, including all inherited members.

Name	Description
“Columns property”	Provides a constant for use with the ULConnection.GetSchema(String) that represents the Columns collection.
“DataSourceInformation property”	Provides a constant for use with the ULConnection.GetSchema(String) that represents the DataSourceInformation collection.
“DataTypes property”	Provides a constant for use with the ULConnection.GetSchema(String) that represents the DataTypes collection.
“ForeignKeys property”	Provides a constant for use with the ULConnection.GetSchema(String) that represents the ForeignKeys collection.
“IndexColumns property”	Provides a constant for use with the ULConnection.GetSchema(String) that represents the IndexColumns collection.
“Indexes property”	Provides a constant for use with the ULConnection.GetSchema(String) that represents the Indexes collection.
“MetaDataCollections property”	Provides a constant for use with the ULConnection.GetSchema(String) that represents the MetaDataCollections collection.
“Publications property”	Provides a constant for use with the ULConnection.GetSchema(String) that represents the Publications collection.
“ReservedWords property”	Provides a constant for use with the ULConnection.GetSchema(String) that represents the ReservedWords collection.

Name	Description
"Restrictions property"	Provides a constant for use with the ULConnection.GetSchema(String) that represents the Restrictions collection.
"Tables property"	Provides a constant for use with the ULConnection.GetSchema(String) that represents the Tables collection.

Columns property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the Columns collection.

Visual Basic syntax

```
Public Shared ReadOnly Property Columns As String
```

C# syntax

```
public string Columns {get;}
```

Remarks

A string representing the name of the Columns collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the Columns collection.

```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.Columns )
```

The following code is the C# language equivalent:

```
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.Columns );
```

DataSourceInformation property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the DataSourceInformation collection.

Visual Basic syntax

```
Public Shared ReadOnly Property DataSourceInformation As String
```


C# syntax

```
public string DataSourceInformation {get;}
```

Remarks

A string representing the name of the DataSourceInformation collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the DataSourceInformation collection.

```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.DataSourceInformation )
```

The following code is the C# language equivalent:

```
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.DataSourceInformation );
```

DataTypes property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the DataTypes collection.

Visual Basic syntax

```
Public Shared ReadOnly Property DataTypes As String
```

C# syntax

```
public string DataTypes {get;}
```

Remarks

A string representing the name of the DataTypes collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the DataTypes collection.

```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.DataTypes )
```

The following code is the C# language equivalent:

```
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.DataTypes );
```

ForeignKeys property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the ForeignKeys collection.

Visual Basic syntax

```
Public Shared ReadOnly Property ForeignKeys As String
```

C# syntax

```
public string ForeignKeys {get;}
```

Remarks

A string representing the name of the ForeignKeys collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the ForeignKeys collection.

```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.ForeignKeys )
```

The following code is the C# language equivalent:

```
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.ForeignKeys );
```

IndexColumns property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the IndexColumns collection.

Visual Basic syntax

```
Public Shared ReadOnly Property IndexColumns As String
```

C# syntax

```
public string IndexColumns {get;}
```

Remarks

A string representing the name of the IndexColumns collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the IndexColumns collection.

```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.IndexColumns )
```

The following code is the C# language equivalent:

```
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.IndexColumns );
```

Indexes property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the Indexes collection.

Visual Basic syntax

```
Public Shared ReadOnly Property Indexes As String
```

C# syntax

```
public string Indexes {get;}
```

Remarks

A string representing the name of the Indexes collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the Indexes collection.

```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.Indexes )
```

The following code is the C# language equivalent:

```
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.Indexes );
```

MetaDataCollections property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the MetaDataCollections collection.

Visual Basic syntax

```
Public Shared ReadOnly Property MetaDataCollections As String
```

C# syntax

```
public string MetaDataCollections {get;}
```

Remarks

A string representing the name of the MetaDataCollections collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the MetaDataCollections collection.

```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.MetaDataCollections )
```

The following code is the C# language equivalent:

```
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.MetaDataCollections );
```

Publications property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the Publications collection.

Visual Basic syntax

```
Public Shared ReadOnly Property Publications As String
```

C# syntax

```
public string Publications {get;}
```

Remarks

A string representing the name of the Publications collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the Publications collection.


```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.Publications )
```

The following code is the C# language equivalent:

```
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.Publications );
```

ReservedWords property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the ReservedWords collection.

Visual Basic syntax

```
Public Shared ReadOnly Property ReservedWords As String
```

C# syntax

```
public string ReservedWords {get;}
```

Remarks

A string representing the name of the ReservedWords collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the ReservedWords collection.

```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.ReservedWords )
```

The following code is the C# language equivalent:

```
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.ReservedWords );
```

Restrictions property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the Restrictions collection.

Visual Basic syntax

```
Public Shared ReadOnly Property Restrictions As String
```

C# syntax

```
public string Restrictions {get;}
```

Remarks

A string representing the name of the Restrictions collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the Restrictions collection.

```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.Restrictions )
```

The following code is the C# language equivalent:

```
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.Restrictions );
```

Tables property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the Tables collection.

Visual Basic syntax

```
Public Shared ReadOnly Property Tables As String
```

C# syntax

```
public string Tables {get;}
```

Remarks

A string representing the name of the Tables collection.

See also

- "GetSchema method" on page 133

Example

The following code fills a DataTable with the Tables collection.

```
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.Tables )


// C#
```



```
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.Tables );
```

ULParameter class

Represents a parameter to a ULCommand.

Visual Basic syntax

```
Public NotInheritable Class ULParameter
    Inherits System.Data.Common.DbParameter
    Implements System.ICloneable
```

C# syntax

```
public sealed class ULParameter :
    System.Data.Common.DbParameter,
    System.ICloneable
```

Base classes

- System.Data.Common.DbParameter
- System.ICloneable

Members

All members of ULParameter class, including all inherited members.

Name	Description
"ULParameter constructor"	Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.
"ResetDbType method"	This method is not supported in UltraLite.NET.
"ToString method"	Returns the string representation of this instance.
"DbType property"	Specifies the System.Data.DbType of the parameter
"Direction property"	A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.
"IsNullable property"	Specifies whether the parameter accepts null values.
"Offset property"	Specifies the offset to the ULParameter.Value.
"ParameterName property"	Specifies the name of the parameter.
"Precision property"	Specifies the maximum number of digits used to represent the ULParameter.Value.

Name	Description
“Scale property”	Specifies the number of decimal places to which ULParameter.Value is resolved.
“Size property”	Specifies the maximum size of the data within the column.
“SourceColumn property”	Specifies the name of the source column mapped to the DataSet and used for loading or returning the value.
“SourceColumnNullMapping property”	Specifies whether the source column is nullable.
“SourceVersion property”	The System.Data.DataRowVersion to use when loading ULParameter.Value.
“ULDbType property”	Specifies the iAnywhere.Data.UltraLite.ULDbType of the parameter
“Value property”	Specifies the value of the parameter.

Remarks

A ULParameter object can be created directly using one of its many constructors, or using the ULCommand.CreateParameter method. Because of the special treatment of the 0 and 0.0 constants and the way overloaded methods are resolved, it is highly recommended that you explicitly cast constant values to type object when using the ULParameter(string,object) constructor. For example:

```
' Visual Basic
Dim p As ULParameter = New ULParameter( "", CType( 0, Object ) )
```

The following code is the C# language equivalent:

```
// C#
ULParameter p = new ULParameter( "", (object)0 );
```

Parameters (including those created by ULCommand.CreateParameter) must be added to a ULCommand.Parameters collection to be used. All parameters are treated as positional parameters and are used by a command in the order that they were added.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- “ULCommand class” on page 64
- “CreateParameter method” on page 77
- “ULParameter constructor” on page 303
- “Parameters property” on page 97
- “Value property” on page 315
- System.Data.Common.DbParameter

ULParameter constructor

Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.

Overload list

Name	Description
"ULParameter() constructor"	Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.
"ULParameter(string, object) constructor"	Initializes a ULParameter object with the specified parameter name and value.
"ULParameter(string, ULDbType) constructor"	Initializes a ULParameter object with the specified parameter name and data type.
"ULParameter(string, ULDbType, int) constructor"	Initializes a ULParameter object with the specified parameter name and data type.
"ULParameter(string, ULDbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object) constructor"	Initializes a ULParameter object with the specified parameter name, data type, length, direction, nullability, numeric precision, numeric scale, source column, source version, and value.
"ULParameter(string, ULDbType, int, string) constructor"	Initializes a ULParameter object with the specified parameter name, data type, and length.

ULParameter() constructor

Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ULParameter()
```

See also

- "Value property" on page 315
- "ULParameter constructor" on page 303
- "ULCommand class" on page 64

Example

The following code creates a ULParameter with the value 3 and adds it to a ULCommand called cmd.

```
' Visual Basic
Dim p As ULParameter = New ULParameter
p.Value = 3
cmd.Parameters.Add( p )
```

The following code is the C# language equivalent:

```
// C#
ULParameter p = new ULParameter();
p.Value = 3;
cmd.Parameters.Add( p );
```

ULParameter(string, object) constructor

Initializes a ULParameter object with the specified parameter name and value.

Visual Basic syntax

```
Public Sub New(ByVal parameterName As String, ByVal value As Object)
```

C# syntax

```
public ULParameter(string parameterName, object value)
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand.
- **value** A System.Object that is to be the value of the parameter.

Remarks

Because of the special treatment of the 0 and 0.0 constants and the way overloaded methods are resolved, it is highly recommended that you explicitly cast constant values to type object when using this constructor.

See also

- "ULParameter constructor" on page 303
- "ULCommand class" on page 64
- System.Object

Example

The following code creates a ULParameter with the value 0 and adds it to a ULCommand called cmd.

```
' Visual Basic
cmd.Parameters.Add( New ULParameter( "", CType( 0, Object ) ) )
```

The following code is the C# language equivalent:

```
// C#
cmd.Parameters.Add( new ULParameter( "", (object)0 ) );
```


ULParameter(string, ULDbType) constructor

Initializes a ULParameter object with the specified parameter name and data type.

Visual Basic syntax

```
Public Sub New(ByVal parameterName As String, ByVal dbType As ULDbType)
```

C# syntax

```
public ULParameter(string parameterName, ULDbType dbType)
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand.

- **dbType** One of the iAnywhere.Data.UltraLite.ULDbType values.

Remarks

This constructor is not recommended; it is provided for compatibility with other data providers.

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "ULParameter constructor" on page 303
- "Value property" on page 315
- "ULCommand class" on page 64
- "ULDbType enumeration" on page 437

ULParameter(string, ULDbType, int) constructor

Initializes a ULParameter object with the specified parameter name and data type.

Visual Basic syntax

```
Public Sub New(
    ByVal parameterName As String,
    ByVal dbType As ULDbType,
    ByVal size As Integer
)
```

C# syntax

```
public ULParameter(string parameterName, ULDbType dbType, int size)
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand.

- **dbType** One of the iAnywhere.Data.UltraLite.ULDbType values.
- **size** The length of the parameter.

Remarks

This constructor is not recommended; it is provided for compatibility with other data providers.

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "ULParameter constructor" on page 303
- "Value property" on page 315
- "ULDbType enumeration" on page 437

ULParameter(string, ULDbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object) constructor

Initializes a ULParameter object with the specified parameter name, data type, length, direction, nullability, numeric precision, numeric scale, source column, source version, and value.

Visual Basic syntax

```
Public Sub New(
    ByVal parameterName As String,
    ByVal dbType As ULDbType,
    ByVal size As Integer,
    ByVal direction As ParameterDirection,
    ByVal isNullable As Boolean,
    ByVal precision As Byte,
    ByVal scale As Byte,
    ByVal sourceColumn As String,
    ByVal sourceVersion As DataRowVersion,
    ByVal value As Object
)
```

C# syntax

```
public ULParameter(
    string parameterName,
    ULDbType dbType,
    int size,
    ParameterDirection direction,
    bool isNullable,
    byte precision,
    byte scale,
    string sourceColumn,
    DataRowVersion sourceVersion,
    object value
)
```


Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand.
- **dbType** One of the iAnywhere.Data.UltraLite.ULDbType values.
- **size** The length of the parameter.
- **direction** One of the System.Data.ParameterDirection values.
- **isNullable** True if the value of the field can be null; otherwise, false.
- **precision** The total number of digits to the left and right of the decimal point to which Value is resolved.
- **scale** The total number of decimal places to which Value is resolved.
- **sourceColumn** The name of the source column to map.
- **sourceVersion** One of the System.Data.DataRowVersion values.
- **value** An System.Object that is to be the value of the parameter.

Exceptions

- **"ULException class"** Only the System.Data.ParameterDirection.Input direction is supported in UltraLite.NET.

Remarks

This constructor is not recommended; it is provided for compatibility with other data providers.

See also

- "ULParameter constructor" on page 303
- "ULCommand class" on page 64
- "ULDbType enumeration" on page 437
- System.Data.ParameterDirection

ULParameter(string, ULDbType, int, string) constructor

Initializes a ULParameter object with the specified parameter name, data type, and length.

Visual Basic syntax

```
Public Sub New(
    ByVal parameterName As String,
    ByVal dbType As ULDbType,
    ByVal size As Integer,
    ByVal sourceColumn As String
)
```

C# syntax

```
public ULParameter(
    string parameterName,
    ULDbType dbType,
    int size,
    string sourceColumn
)
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand.
- **dbType** One of the iAnywhere.Data.UltraLite.ULDbType values.
- **size** The length of the parameter.
- **sourceColumn** The name of the source column to map.

Remarks

This constructor is not recommended; it is provided for compatibility with other data providers.

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "ULParameter constructor" on page 303
- "Value property" on page 315
- "ULCommand class" on page 64
- "ULDbType enumeration" on page 437

ResetDbType method

This method is not supported in UltraLite.NET.

Visual Basic syntax

```
Public Overrides Sub ResetDbType()
```

C# syntax

```
public override void ResetDbType()
```

Remarks

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "Value property" on page 315

ToString method

Returns the string representation of this instance.

Visual Basic syntax

```
Public Overrides Function ToString() As String
```

C# syntax

```
public override string ToString()
```

Returns

The name of the parameter.

DbType property

Specifies the System.Data.DbType of the parameter

Visual Basic syntax

```
Public Overrides Property DbType As DbType
```

C# syntax

```
public override DbType DbType {get;set;}
```

Exceptions

- **ArgumentException** There is no mapping from the specified value to a iAnywhere.Data.UltraLite.ULDbType, hence, the specified value is not supported.

Remarks

One of the System.Data.DbType values.

The ULParameter.ULDbType and DbType properties are linked. Therefore, setting the DbType changes the ULParameter.ULDbType to a supporting iAnywhere.Data.UltraLite.ULDbType.

See also

- "ULDbType property" on page 314
- "ULDbType enumeration" on page 437
- System.Data.DbType

Direction property

A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

Visual Basic syntax

```
Public Overrides Property Direction As ParameterDirection
```

C# syntax

```
public override ParameterDirection Direction {get;set;}
```

Exceptions

- **"ULException class"** Only the System.Data.ParameterDirection.Input direction is supported in UltraLite.NET.

Remarks

One of the System.Data.ParameterDirection values.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "Value property" on page 315
- System.Data.ParameterDirection

IsNullable property

Specifies whether the parameter accepts null values.

Visual Basic syntax

```
Public Overrides Property IsNullable As Boolean
```

C# syntax

```
public override bool IsNullable {get;set;}
```

Remarks

True if null values are accepted, false otherwise. The default is false. Null values are handled using the DBNull class.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "Value property" on page 315

Offset property

Specifies the offset to the ULParameter.Value.

Visual Basic syntax

```
Public Property Offset As Integer
```

C# syntax

```
public int Offset {get;set;}
```

Remarks

The offset to the value. The default is 0.

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "Value property" on page 315

ParameterName property

Specifies the name of the parameter.

Visual Basic syntax

```
Public Overrides Property ParameterName As String
```

C# syntax

```
public override string ParameterName {get;set;}
```

Remarks

A string representing the name of the parameter, or an empty string ("") for unnamed parameters. Specifying a null reference (Nothing in Visual Basic) results in an empty string being used.

In UltraLite.NET, parameter names are not used by ULCommand. All parameters are treated as positional parameters and are used by a command in the order that they were added.

See also

- "ULCommand class" on page 64

Precision property

Specifies the maximum number of digits used to represent the ULParameter.Value.

Visual Basic syntax

```
Public Property Precision As Byte
```

C# syntax

```
public byte Precision {get;set;}
```

Exceptions

- **ArgumentException** The value is greater then 38.

Remarks

The maximum number of digits used to represent the ULParameter.Value. The default value is 0, which indicates that the data provider sets the precision for the ULParameter.Value.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "Value property" on page 315

Scale property

Specifies the number of decimal places to which ULParameter.Value is resolved.

Visual Basic syntax

```
Public Property Scale As Byte
```

C# syntax

```
public byte Scale {get;set;}
```

Remarks

The number of decimal places to which ULParameter.Value is resolved. The default is 0.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "Value property" on page 315

Size property

Specifies the maximum size of the data within the column.

Visual Basic syntax

```
Public Overrides Property Size As Integer
```

C# syntax

```
public override int Size {get;set;}
```

Remarks

The maximum size of the data within the column. The default value is inferred from the parameter value. The Size property is used for binary and string types.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "Value property" on page 315

SourceColumn property

Specifies the name of the source column mapped to the DataSet and used for loading or returning the value.

Visual Basic syntax

```
Public Overrides Property SourceColumn As String
```

C# syntax

```
public override string SourceColumn {get;set;}
```

Remarks

A string specifying the name of the source column mapped to the DataSet and used for loading or returning the value.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "Value property" on page 315

SourceColumnNullMapping property

Specifies whether the source column is nullable.

Visual Basic syntax

```
Public Overrides Property SourceColumnNullMapping As Boolean
```

C# syntax

```
public override bool SourceColumnNullMapping {get;set;}
```

Remarks

True if the source column is nullable; false, otherwise.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "Value property" on page 315

SourceVersion property

The System.Data.DataRowVersion to use when loading ULParameter.Value.

Visual Basic syntax

```
Public Overrides Property SourceVersion As DataRowVersion
```

C# syntax

```
public override DataRowVersion SourceVersion {get;set;}
```

See also

- "Value property" on page 315
- System.Data.DataRowVersion

ULDbType property

Specifies the iAnywhere.Data.UltraLite.ULDbType of the parameter

Visual Basic syntax

```
Public Property ULDbType As ULDbType
```

C# syntax

```
public ULDbType ULDbType {get;set;}
```

Remarks

One of the iAnywhere.Data.UltraLite.ULDbType values.

The ULDbType and ULParameter.DbType are linked. Therefore, setting the ULDbType changes the ULParameter.DbType to a supporting System.Data.DbType.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value is important.

See also

- "Value property" on page 315
- "DbType property" on page 309
- "ULDbType enumeration" on page 437
- System.Data.DbType

Value property

Specifies the value of the parameter.

Visual Basic syntax

```
Public Overrides Property Value As Object
```

C# syntax

```
public override object Value {get;set;}
```

Remarks

A System.Object that specifies the value of the parameter.

The value is sent as-is to the data provider without any type conversion or mapping. When the command is executed, the command attempts to convert the value to the required type, signaling a ULException with ULSQLCode.SQLE_CONVERSION_ERROR if it cannot convert the value.

See also

- "ULException class" on page 257
- System.Object

ULParameterCollection class

Represents all parameters to a ULCommand.

Visual Basic syntax

```
Public NotInheritable Class ULParameterCollection
    Inherits System.Data.Common.DbParameterCollection
```

C# syntax

```
public sealed class ULParameterCollection :
    System.Data.Common.DbParameterCollection
```

Base classes

- System.Data.Common.DbParameterCollection

Members

All members of ULParameterCollection class, including all inherited members.

Name	Description
"Add method"	Adds a ULParameter to the collection.
"AddRange method"	Adds an array of values to the end of the ULParameterCollection.
"Clear method"	Removes all the parameters from the collection.
"Contains method"	Checks whether a ULParameter exists in the collection.
"CopyTo method"	Copies ULParameter objects from the ULParameterCollection to the specified array.
"GetEnumerator method"	Returns an enumerator for the collection.
GetParameter method (Inherited from System.Data.Common.DbParameterCollection)	Returns the System.Data.Common.DbParameter object at the specified index in the collection.
"IndexOf method"	Returns the location of the ULParameter in the collection.
"Insert method"	Inserts an ULParameter in the collection at the specified index.
"Remove method"	Removes an ULParameter from the collection.
"RemoveAt method"	Removes the parameter at the specified index in the collection.
SetParameter method (Inherited from System.Data.Common.DbParameterCollection)	Sets the System.Data.Common.DbParameter object at the specified index to a new value.

Name	Description
"Count property"	Returns the number of ULParameter objects in the collection.
"IsFixedSize property"	Indicates whether the ULParameterCollection has a fixed size.
"IsReadOnly property"	Indicates whether the ULParameterCollection is read-only.
"IsSynchronized property"	Indicates whether the ULParameterCollection is synchronized.
"SyncRoot property"	Returns an object that can be used to synchronize access to the SAParameterCollection.
"this property"	Returns the ULParameter at the specified index.

Remarks

All parameters in the collection are treated as positional parameters and are specified in the same order as the question mark placeholders in the ULCommand.CommandText. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText as there are parameters in the collection. Nulls are substituted for missing parameters.

There is no constructor for ULParameterCollection. You obtain a ULParameterCollection from the ULCommand.Parameters.

See also

- "ULCommand class" on page 64
- "CommandText property" on page 94
- "Parameters property" on page 97
- System.Data.Common.DbParameterCollection

Add method

Adds a ULParameter to the collection.

Overload list

Name	Description
"Add(object) method"	Adds a ULParameter to the collection.
"Add(string, object) method"	Adds a new ULParameter, created using the specified parameter name and value, to the collection.
"Add(string, ULDbType) method"	Adds a new ULParameter, created using the specified parameter name and data type, to the collection.
"Add(string, ULDbType, int) method"	Adds a new ULParameter, created using the specified parameter name, data type, and length, to the collection.
"Add(string, ULDbType, int, string) method"	Adds a new ULParameter, created using the specified parameter name, data type, length, and source column name, to the collection.
"Add(ULParameter) method"	Adds a ULParameter to the collection.

Add(object) method

Adds a ULParameter to the collection.

Visual Basic syntax

```
Public Overrides Function Add(ByVal value As Object) As Integer
```

C# syntax

```
public override int Add(object value)
```

Parameters

- **value** The ULParameter object to add to the collection.

Returns

The index of the new ULParameter object.

Exceptions

- **ArgumentNullException** The value cannot be null (Nothing in Visual Basic).
- **InvalidCastException** The value specified must be a ULParameter.
- **ArgumentException** The ULParameter object can only be added to the collection once.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText. For

example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "Add method" on page 317
- "CommandText property" on page 94
- "ULParameter class" on page 301

Add(string, object) method

Adds a new ULParameter, created using the specified parameter name and value, to the collection.

Visual Basic syntax

```
Public Function Add(
    ByVal parameterName As String,
    ByVal value As Object
) As ULParameter
```

C# syntax

```
public ULParameter Add(string parameterName, object value)
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand.

- **value** A System.Object that is to be the value of the parameter.

Returns

The new ULParameter object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText as there are parameters in the collection. Nulls are substituted for missing parameters.

Because of the special treatment of the 0 and 0.0 constants and the way overloaded methods are resolved, it is highly recommended that you explicitly cast constant values to type object when using this method.

See also

- "Add method" on page 317
- "ULParameter class" on page 301
- "CommandText property" on page 94
- "ULCommand class" on page 64
- System.Object

Example

The following code adds a ULParameter with the value 0 to a ULCommand called cmd.

```
' Visual Basic
cmd.Parameters.Add( "", CType( 0, Object ) )
```

The following code is the C# language equivalent:

```
// C#
cmd.Parameters.Add( "", (object)0 );
```

Add(string, ULDbType) method

Adds a new ULParameter, created using the specified parameter name and data type, to the collection.

Visual Basic syntax

```
Public Function Add(
    ByVal parameterName As String,
    ByVal ulDbType As ULDbType
) As ULParameter
```

C# syntax

```
public ULParameter Add(string parameterName, ULDbType ulDbType)
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand.
- **ulDbType** One of the iAnywhere.Data.UltraLite.ULDbType values.

Returns

The new ULParameter object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "Add method" on page 317
- "ULParameter class" on page 301
- "CommandText property" on page 94
- "ULCommand class" on page 64
- "ULDbType enumeration" on page 437

Add(string, ULDbType, int) method

Adds a new ULParameter, created using the specified parameter name, data type, and length, to the collection.

Visual Basic syntax

```
Public Function Add(
    ByVal parameterName As String,
    ByVal ulDbType As ULDbType,
    ByVal size As Integer
) As ULParameter
```

C# syntax

```
public ULParameter Add(
    string parameterName,
    ULDbType ulDbType,
    int size
)
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand.
- **ulDbType** One of the iAnywhere.Data.UltraLite.ULDbType values.
- **size** The length of the parameter.

Returns

The new ULParameter object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "Add method" on page 317
- "ULParameter class" on page 301
- "CommandText property" on page 94
- "ULCommand class" on page 64
- "ULDbType enumeration" on page 437

Add(string, ULDbType, int, string) method

Adds a new ULParameter, created using the specified parameter name, data type, length, and source column name, to the collection.

Visual Basic syntax

```
Public Function Add(
    ByVal parameterName As String,
    ByVal ulDbType As ULDbType,
    ByVal size As Integer,
    ByVal sourceColumn As String
) As ULParameter
```

C# syntax

```
public ULParameter Add(
    string parameterName,
    ULDbType ulDbType,
    int size,
    string sourceColumn
)
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand.
- **ulDbType** One of the iAnywhere.Data.UltraLite.ULDbType values.
- **size** The length of the parameter.
- **sourceColumn** The name of the source column to map.

Returns

The new ULParameter object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and

so on. There must be at least as many question marks in the ULCommand.CommandText as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "Add method" on page 317
- "ULParameter class" on page 301
- "CommandText property" on page 94
- "ULCommand class" on page 64
- "ULDbType enumeration" on page 437

Add(ULParameter) method

Adds a ULParameter to the collection.

Visual Basic syntax

```
Public Function Add(ByVal value As ULParameter) As ULParameter
```

C# syntax

```
public ULParameter Add(ULParameter value)
```

Parameters

- **value** The ULParameter object to add to the collection.

Returns

The new ULParameter object.

Exceptions

- **ArgumentNullException** The value cannot be null (Nothing in Visual Basic).
- **ArgumentException** The ULParameter object can only be added to the collection once.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "Add method" on page 317
- "ULParameter class" on page 301
- "CommandText property" on page 94

AddRange method

Adds an array of values to the end of the ULParameterCollection.

Overload list

Name	Description
"AddRange(Array) method"	Adds an array of values to the end of the ULParameterCollection.
"AddRange(ULParameter[]) method"	Adds an array of values to the end of the ULParameterCollection.

AddRange(Array) method

Adds an array of values to the end of the ULParameterCollection.

Visual Basic syntax

```
Public Overrides Sub AddRange(ByVal values As Array)
```

C# syntax

```
public override void AddRange(Array values)
```

Parameters

- **values** An array of ULParameter objects to add to the end of this collection.

See also

- "ULParameter class" on page 301

AddRange(ULParameter[]) method

Adds an array of values to the end of the ULParameterCollection.

Visual Basic syntax

```
Public Sub AddRange(ByVal values As ULParameter())
```

C# syntax

```
public void AddRange(ULParameter[] values)
```

Parameters

- **values** An array of ULParameter objects to add to the end of this collection.

Remarks

This is the strongly-typed version of DbParameterCollection.AddRange(Array).

See also

- "ULParameter class" on page 301
- "ULParameterCollection class" on page 315

Clear method

Removes all the parameters from the collection.

Visual Basic syntax

```
Public Overrides Sub Clear()
```

C# syntax

```
public override void Clear()
```

Contains method

Checks whether a ULParameter exists in the collection.

Overload list

Name	Description
"Contains(object) method"	Checks whether a ULParameter exists in the collection.
"Contains(string) method"	Checks whether a ULParameter with the specified name exists in the collection.

Contains(object) method

Checks whether a ULParameter exists in the collection.

Visual Basic syntax

```
Public Overrides Function Contains(ByVal value As Object) As Boolean
```

C# syntax

```
public override bool Contains(object value)
```

Parameters

- **value** The ULParameter object to check for.

Returns

True if the collection contains the ULParameter, false otherwise.

See also

- "Contains method" on page 325
- "ULParameter class" on page 301

Contains(string) method

Checks whether a ULParameter with the specified name exists in the collection.

Visual Basic syntax

```
Public Overrides Function Contains(ByVal value As String) As Boolean
```

C# syntax

```
public override bool Contains(string value)
```

Parameters

- **value** The name of the parameter to search for.

Returns

True if the collection contains the ULParameter, false otherwise.

See also

- "Contains method" on page 325
- "ULParameter class" on page 301

CopyTo method

Copies ULParameter objects from the ULParameterCollection to the specified array.

Visual Basic syntax

```
Public Overrides Sub CopyTo(
    ByVal array As Array,
    ByVal index As Integer
)
```

C# syntax

```
public override void CopyTo(Array array, int index)
```

Parameters

- **array** The array into which to copy the ULParameter objects.
- **index** The starting index of the array.

See also

- "ULParameter class" on page 301

GetEnumerator method

Returns an enumerator for the collection.

Visual Basic syntax

```
Public Overrides Function GetEnumerator()
    As System.Collections.IEnumerator
```

C# syntax

```
public override IEnumerator GetEnumerator()
```

Returns

An ArrayList enumerator enumerating the parameters in the collection.

IndexOf method

Returns the location of the ULParameter in the collection.

Overload list

Name	Description
"IndexOf(object) method"	Returns the location of the ULParameter in the collection.
"IndexOf(string) method"	Returns the location of the ULParameter with the specified name in the collection.

IndexOf(object) method

Returns the location of the ULParameter in the collection.

Visual Basic syntax

```
Public Overrides Function IndexOf(ByVal value As Object) As Integer
```

C# syntax

```
public override int IndexOf(object value)
```

Parameters

- **value** The ULParameter object to locate.

Returns

The zero-based index of the ULParameter in the collection or -1 if the parameter is not found.

Exceptions

- **InvalidCastException** The value specified must be a ULParameter.

See also

- "IndexOf method" on page 327
- "ULParameter class" on page 301

IndexOf(string) method

Returns the location of the ULParameter with the specified name in the collection.

Visual Basic syntax

```
Public Overrides Function IndexOf(
    ByVal parameterName As String
) As Integer
```

C# syntax

```
public override int IndexOf(string parameterName)
```

Parameters

- **parameterName** The name of the parameter to locate.

Returns

The zero-based index of the ULParameter in the collection or -1 if the parameter is not found.

See also

- "IndexOf method" on page 327
- "ULParameter class" on page 301

Insert method

Inserts an ULParameter in the collection at the specified index.

Visual Basic syntax

```
Public Overrides Sub Insert(
    ByVal index As Integer,
    ByVal value As Object
)
```

C# syntax

```
public override void Insert(int index, object value)
```

Parameters

- **index** The zero-based index where the parameter is to be inserted within the collection.

- **value** The ULParameter object to insert.

Exceptions

- **IndexOutOfRangeException** The index is invalid.
- **ArgumentNullException** You cannot set a parameter using a null reference (Nothing in Visual Basic).
- **InvalidCastException** The value specified must be a ULParameter.

See also

- "ULParameter class" on page 301

Remove method

Removes an ULParameter from the collection.

Visual Basic syntax

```
Public Overrides Sub Remove(ByVal value As Object)
```

C# syntax

```
public override void Remove(object value)
```

Parameters

- **value** The ULParameter object to remove.

Exceptions

- **ArgumentNullException** You cannot set a parameter using a null reference (Nothing in Visual Basic).
- **InvalidCastException** The value specified must be a ULParameter.
- **ArgumentException** The collection does not contain the specified parameter.

See also

- "ULParameter class" on page 301

RemoveAt method

Removes the parameter at the specified index in the collection.

Overload list

Name	Description
"RemoveAt(int) method"	Removes the parameter at the specified index in the collection.
"RemoveAt(string) method"	Removes the parameter with the specified name from the collection.

RemoveAt(int) method

Removes the parameter at the specified index in the collection.

Visual Basic syntax

```
Public Overrides Sub RemoveAt(ByVal index As Integer)
```

C# syntax

```
public override void RemoveAt(int index)
```

Parameters

- **index** The zero-based index of the parameter to remove. The value must be in the range [0,ULParameterCollection.Count-1]. The first parameter in the collection has an index value of zero.

Exceptions

- **IndexOutOfRangeException** The index is invalid.

See also

- "RemoveAt method" on page 329
- "Count property" on page 331

RemoveAt(string) method

Removes the parameter with the specified name from the collection.

Visual Basic syntax

```
Public Overrides Sub RemoveAt(ByVal parameterName As String)
```

C# syntax

```
public override void RemoveAt(string parameterName)
```

Parameters

- **parameterName** The name of the parameter to retrieve.

Exceptions

- **IndexOutOfRangeException** There is no parameter with the specified name.

See also

- "RemoveAt method" on page 329

Count property

Returns the number of ULParameter objects in the collection.

Visual Basic syntax

```
Public ReadOnly Overrides Property Count As Integer
```

C# syntax

```
public override int Count {get;}
```

Remarks

The number of ULParameter objects in the collection.

IsFixedSize property

Indicates whether the ULParameterCollection has a fixed size.

Visual Basic syntax

```
Public ReadOnly Overrides Property IsFixedSize As Boolean
```

C# syntax

```
public override bool IsFixedSize {get;}
```

Remarks

True if this collection has a fixed size, false otherwise.

IsReadOnly property

Indicates whether the ULParameterCollection is read-only.

Visual Basic syntax

```
Public ReadOnly Overrides Property IsReadOnly As Boolean
```

C# syntax

```
public override bool IsReadOnly {get;}
```

Remarks

True if this collection is read-only, false otherwise.

IsSynchronized property

Indicates whether the ULParameterCollection is synchronized.

Visual Basic syntax

```
Public ReadOnly Overrides Property IsSynchronized As Boolean
```

C# syntax

```
public override bool IsSynchronized {get;}
```

Remarks

True if this collection is synchronized, false otherwise.

SyncRoot property

Returns an object that can be used to synchronize access to the SAParameterCollection.

Visual Basic syntax

```
Public ReadOnly Overrides Property SyncRoot As Object
```

C# syntax

```
public override object SyncRoot {get;}
```

Remarks

The object to be used to synchronize access to this collection.

this property

Returns the ULParameter at the specified index.

Overload list

Name	Description
“this[int] property”	Returns the ULParameter at the specified index.
“this[string] property”	Returns the ULParameter with the specified name.

this[int] property

Returns the ULParameter at the specified index.

Visual Basic syntax

```
Public Shadows Property Item(ByVal index As Integer) As ULParameter
```

C# syntax

```
public new ULParameter this[int index] {get;set;}
```

Parameters

- **index** The zero-based index of the parameter to retrieve. The value must be in the range [0,ULParameterCollection.Count-1]. The first parameter in the collection has an index value of zero.

Returns

The ULParameter at the specified index.

Exceptions

- **IndexOutOfRangeException** The index is invalid.

Remarks

In C#, this property is the indexer for the ULParameterCollection class.

This is the strongly-typed version of DbParameterCollection.this[int].

See also

- "ULParameter class" on page 301

this[string] property

Returns the ULParameter with the specified name.

Visual Basic syntax

```
Public Shadows Property Item(
    ByVal parameterName As String
) As ULParameter
```

C# syntax

```
public new ULParameter this[string parameterName] {get;set;}
```

Parameters

- **parameterName** The name of the parameter to retrieve.

Returns

The ULParameter with the specified name.

Exceptions

- **IndexOutOfRangeException** There is no parameter with the specified name.

- **ArgumentNullException** You cannot set a parameter using a null (Nothing in Visual Basic) parameter name.

Remarks

In C#, this property is the indexer for the ULParameterCollection class.

This is the strongly-typed version of DbParameterCollection.this[string].

See also

- "GetOrdinal method" on page 239
- "GetValue method" on page 246
- "GetFieldType method" on page 235
- "ULParameter class" on page 301

ULResultSet class

UL Ext: Represents an editable result set in an UltraLite database.

Visual Basic syntax

```
Public Class ULResultSet Inherits ULDataReader
```

C# syntax

```
public class ULResultSet : ULDataReader
```

Base classes

- "ULDataReader class" on page 220

Derived classes

- "ULTable class" on page 394

Members

All members of ULResultSet class, including all inherited members.

Name	Description
"AppendBytes method"	Appends the specified subset of the specified array of System.Bytes to the new value for the specified ULDbType.LongBinary column.

Name	Description
"AppendChars method"	Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType.LongVarchar column.
"Close method"	Closes the cursor.
"Delete method"	Deletes the current row.
Dispose method (Inherited from System.Data.Common.DbDataReader)	Releases all resources used by the current instance of the System.Data.Common.DbDataReader class.
"GetBoolean method"	Returns the value for the specified column as a System.Boolean.
"GetByte method"	Returns the value for the specified column as an unsigned 8-bit value (System.Byte).
"GetBytes method"	**UL Ext:** Returns the value for the specified column as an array of System.Bytes.
"GetChar method"	This method is not supported in UltraLite.NET.
"GetChars method"	Copies a subset of the value for the specified ULDbType.LongVarchar column, beginning at the specified offset, to the specified offset of the destination System.Char array.
GetData method (Inherited from System.Data.Common.DbDataReader)	Returns a System.Data.Common.DbDataReader object for the requested column ordinal.
"GetDataTypeName method"	Returns the name of the specified column's provider data type.
"GetDateTime method"	Returns the value for the specified column as a System.DateTime with millisecond accuracy.

Name	Description
GetDbDataReader method (Inherited from System.Data.Common.DbDataReader)	Returns a System.Data.Common.DbDataReader object for the requested column ordinal that can be overridden with a provider-specific implementation.
"GetDecimal method"	Returns the value for the specified column as a System.Decimal.
"GetDouble method"	Returns the value for the specified column as a System.Double.
"GetEnumerator method"	Returns an System.Collections.IEnumerator that iterates through the ULDataReader.
"GetFieldType method"	Returns the System.Type most appropriate for the specified column.
"GetFloat method"	Returns the value for the specified column as a System.Single.
"GetGuid method"	Returns the value for the specified column as a UUID (System.Guid).
"GetInt16 method"	Returns the value for the specified column as a System.Int16.
"GetInt32 method"	Returns the value for the specified column as an Int32.
"GetInt64 method"	Returns the value for the specified column as an Int64.
"GetName method"	Returns the name of the specified column.
"GetOrdinal method"	Returns the column ID of the named column.
GetProviderSpecificFieldType method (Inherited from System.Data.Common.DbDataReader)	Returns the provider-specific field type of the specified column.

Name	Description
GetProviderSpecificValue method (Inherited from System.Data.Common.DbDataReader)	Gets the value of the specified column as an instance of System.Object.
GetProviderSpecificValues method (Inherited from System.Data.Common.DbDataReader)	Gets all provider-specific attribute columns in the collection for the current row.
"GetRowCount method"	**UL Ext:** Returns the number of rows in the cursor, within threshold.
"GetSchemaTable method"	Returns a System.Data.DataTable that describes the column metadata of the ULDataReader.
"GetString method"	Returns the value for the specified column as a System.String.
"GetTimeSpan method"	Returns the value for the specified column as a System.TimeSpan with millisecond accuracy.
"GetUInt16 method"	Returns the value for the specified column as a System.UInt16.
"GetUInt32 method"	Returns the value for the specified column as a UInt32.
"GetUInt64 method"	Returns the value for the specified column as a System.UInt64.
"GetValue method"	Returns the value of the specified column in its native format.
"GetValues method"	Returns all the column values for the current row.
"IsDBNull method"	Checks whether the value from the specified column is NULL.
"MoveAfterLast method"	**UL Ext:** Positions the cursor to after the last row of the cursor.
"MoveBeforeFirst method"	**UL Ext:** Positions the cursor to before the first row of the cursor.

Name	Description
“MoveFirst method”	**UL Ext:** Positions the cursor to the first row of the cursor.
“MoveLast method”	**UL Ext:** Positions the cursor to the last row of the cursor.
“MoveNext method”	**UL Ext:** Positions the cursor to the next row or after the last row if the cursor was already on the last row.
“MovePrevious method”	**UL Ext:** Positions the cursor to the previous row or before the first row.
“MoveRelative method”	**UL Ext:** Positions the cursor relative to the current row.
“NextResult method”	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
“Read method”	Positions the cursor to the next row, or after the last row if the cursor was already on the last row.
“SetBoolean method”	Sets the value for the specified column using a System.Boolean.
“SetByte method”	Sets the value for the specified column using a System.Byte (unsigned 8-bit integer).
“SetBytes method”	Sets the value for the specified column using an array of System.Bytes.
“SetDateTime method”	Sets the value for the specified column using a System.DateTime.
“SetDBNull method”	Sets a column to NULL.
“SetDecimal method”	Sets the value for the specified column using a System.Decimal.

Name	Description
"SetDouble method"	Sets the value for the specified column using a System.Double.
"SetFloat method"	Sets the value for the specified column using a System.Single.
"SetGuid method"	Sets the value for the specified column using a System.Guid.
"SetInt16 method"	Sets the value for the specified column using an System.Int16.
"SetInt32 method"	Sets the value for the specified column using an System.Int32.
"SetInt64 method"	Sets the value for the specified column using an Int64.
"SetString method"	Sets the value for the specified column using a System.String.
"SetTimeSpan method"	Sets the value for the specified column using a System.TimeSpan.
"SetToDefault method"	Sets the value for the specified column to its default value.
"SetUInt16 method"	Sets the value for the specified column using a System.UInt16.
"SetUInt32 method"	Sets the value for the specified column using an System.UInt32.
"SetUInt64 method"	Sets the value for the specified column using a System.UInt64.
"Update method"	Updates the current row with the current column values (specified using the set methods).
"UpdateBegin method"	Prepares to update the current row.
"Depth property"	Returns the depth of nesting for the current row.
"FieldCount property"	Returns the number of columns in the cursor.

Name	Description
"HasRows property"	Checks whether the ULDataReader has one or more rows.
"IsBOF property"	**UL Ext:** Checks whether the current row position is before the first row.
"IsClosed property"	Checks whether the cursor is currently open.
"IsEOF property"	**UL Ext:** Checks whether the current row position is after the last row.
"RecordsAffected property"	Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.
"RowCount property"	**UL Ext:** Returns the number of rows in the cursor.
"Schema property"	**UL Ext:** Holds the schema of this cursor.
"this property"	Returns the value of the specified column in its native format.
VisibleFieldCount property (Inherited from System.Data.Common.DbDataReader)	Gets the number of fields in the System.Data.Common.DbDataReader that are not hidden.

Remarks

There is no constructor for this class. ResultSets are created using the ULCommand.ExecuteResultSet() method of the ULCommand class.

```
' Visual Basic
Dim cmd As ULCommand = new ULCommand( _
    "SELECT emp_id FROM employee", conn _
)
Dim resultSet As ULResultSet = cmd.ExecuteResultSet()
```

The following code is the C# language equivalent:

```
// C#
ULCommand cmd = new ULCommand(
    "SELECT emp_id FROM employee", conn
);
ULResultSet resultSet = cmd.ExecuteResultSet();
```


A ULResultSet object represents an editable result set on which you can perform positioned updates and deletes. For fully editable result sets, use ULCommand.ExecuteTable() or a ULDataAdapter.

See also

- "ExecuteResultSet method" on page 87
- "ULCommand class" on page 64
- "ULResultSet class" on page 334
- "ExecuteTable method" on page 91
- "ULDataAdapter class" on page 196
- "ULDataReader class" on page 220
- System.Data.IDataReader

AppendBytes method

Appends the specified subset of the specified array of System.Bytes to the new value for the specified ULDbType.LongBinary column.

Visual Basic syntax

```
Public Sub AppendBytes(
    ByVal colID As Integer,
    ByVal val As Byte(),
    ByVal srcOffset As Integer,
    ByVal count As Integer
)
```

C# syntax

```
public void AppendBytes(int colID, byte[] val, int srcOffset, int count)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** 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.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The bytes at position *srcOffset* (starting from 0) through *srcOffset* +*count* -1 of the array *val* are appended to the value for the specified column.

When inserting, ULTable.InsertBegin initializes the new value to the column's default value. The data in the row is not actually changed until you execute an ULTable.Insert, and that change is not made permanent until it is committed.

When updating, the first append on a column clears the current value prior to appending the new value.

If any of the following are true, a ULException with code ULSQLCode.SQLE_INVALID_PARAMETER is thrown and the destination is not modified:

- *val* is null.
- *srcOffset* is negative.
- *count* is negative.
- *srcOffset* +*count* is greater than the *val* length.

For other errors, a ULException with the appropriate error code is thrown.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "InsertBegin method" on page 411
- "Insert method" on page 411
- "ULException class" on page 257
- "FieldCount property" on page 252
- System.Byte

AppendChars method

Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType.LongVarchar column.

Visual Basic syntax

```
Public Sub AppendChars(
    ByVal colID As Integer,
    ByVal val As Char(),
    ByVal srcOffset As Integer,
    ByVal count As Integer
)
```

C# syntax

```
public void AppendChars(int colID, char[] val, int srcOffset, int count)
```


Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** 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.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The characters at position *srcOffset* (starting from 0) through *srcOffset* +*count* -1 of the array *val* are appended to the value for the specified column. When inserting, ULTable.InsertBegin initializes the new value to the column's default value. The data in the row is not actually changed until you execute an ULTable.Insert, and that change is not made permanent until it is committed.

When updating, the first append on a column clears the current value prior to appending the new value.

If any of the following is true, a ULException with code ULSQLCode.SQLE_INVALID_PARAMETER is thrown and the destination is not modified:

- *val* is null.
- *srcOffset* is negative.
- *count* is negative.
- *srcOffset* +*count* is greater than *value* length.

For other errors, a ULException with the appropriate error code is thrown.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "InsertBegin method" on page 411
- "Insert method" on page 411
- "ULException class" on page 257
- "FieldCount property" on page 252
- System.Char

Delete method

Deletes the current row.

Visual Basic syntax

```
Public Sub Delete()
```

C# syntax

```
public void Delete()
```

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "StartSynchronizationDelete method" on page 140
- "StopSynchronizationDelete method" on page 141

SetBoolean method

Sets the value for the specified column using a System.Boolean.

Visual Basic syntax

```
Public Sub SetBoolean(ByVal colID As Integer, ByVal val As Boolean)
```

C# syntax

```
public void SetBoolean(int colID, bool val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.Boolean

SetByte method

Sets the value for the specified column using a System.Byte (unsigned 8-bit integer).

Visual Basic syntax

```
Public Sub SetByte(ByVal colID As Integer, ByVal val As Byte)
```

C# syntax

```
public void SetByte(int colID, byte val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.Byte

SetBytes method

Sets the value for the specified column using an array of System.Bytes.

Visual Basic syntax

```
Public Sub SetBytes(ByVal colID As Integer, ByVal val As Byte())
```

C# syntax

```
public void SetBytes(int colID, byte[] val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Only suitable for columns of type ULDbType.Binary or ULDbType.LongBinary, or for columns of type ULDbType.UniqueIdentifier when val is of length 16. The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.Byte

SetDateTime method

Sets the value for the specified column using a System.DateTime.

Visual Basic syntax

```
Public Sub SetDateTime(ByVal colID As Integer, ByVal val As Date)
```

C# syntax

```
public void SetDateTime(int colID, DateTime val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The set value is accurate to the millisecond. The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.DateTime

SetDBNull method

Sets a column to NULL.

Visual Basic syntax

```
Public Sub SetDBNull(ByVal colID As Integer)
```

C# syntax

```
public void SetDBNull(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "IsColumnNullable method" on page 427
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252

SetDecimal method

Sets the value for the specified column using a System.Decimal.

Visual Basic syntax

```
Public Sub SetDecimal(ByVal colID As Integer, ByVal val As Decimal)
```

C# syntax

```
public void SetDecimal(int colID, decimal val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.Decimal

SetDouble method

Sets the value for the specified column using a System.Double.

Visual Basic syntax

```
Public Sub SetDouble(ByVal colID As Integer, ByVal val As Double)
```

C# syntax

```
public void SetDouble(int colID, double val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.Double

SetFloat method

Sets the value for the specified column using a System.Single.

Visual Basic syntax

```
Public Sub SetFloat(ByVal colID As Integer, ByVal val As Single)
```

C# syntax

```
public void SetFloat(int colID, float val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- “GetOrdinal method” on page 239
- “Schema property” on page 255
- “GetFieldType method” on page 235
- “Insert method” on page 411
- “Update method” on page 357
- “FieldCount property” on page 252
- System.Single

SetGuid method

Sets the value for the specified column using a System.Guid.

Visual Basic syntax

```
Public Sub SetGuid(ByVal colID As Integer, ByVal val As Guid)
```

C# syntax

```
public void SetGuid(int colID, Guid val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **“ULException class”** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed. Only valid for columns of type ULDbType.UniqueIdentifier or for columns of type ULDbType.Binary with length 16.

See also

- “GetNewUUID method” on page 130
- “GetOrdinal method” on page 239
- “Schema property” on page 255
- “GetFieldType method” on page 235
- “GetColumnSize method” on page 193
- “Insert method” on page 411
- “Update method” on page 357
- “FieldCount property” on page 252
- System.Guid

SetInt16 method

Sets the value for the specified column using an System.Int16.

Visual Basic syntax

```
Public Sub SetInt16(ByVal colID As Integer, ByVal val As Short)
```

C# syntax

```
public void SetInt16(int colID, short val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.Int16

SetInt32 method

Sets the value for the specified column using an System.Int32.

Visual Basic syntax

```
Public Sub SetInt32(ByVal colID As Integer, ByVal val As Integer)
```

C# syntax

```
public void SetInt32(int colID, int val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.Int32

SetInt64 method

Sets the value for the specified column using an Int64.

Visual Basic syntax

```
Public Sub SetInt64(ByVal colID As Integer, ByVal val As Long)
```

C# syntax

```
public void SetInt64(int colID, long val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.Int64

SetString method

Sets the value for the specified column using a System.String.

Visual Basic syntax

```
Public Sub SetString(ByVal colID As Integer, ByVal val As String)
```

C# syntax

```
public void SetString(int colID, string val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252

SetTimeSpan method

Sets the value for the specified column using a System.TimeSpan.

Visual Basic syntax

```
Public Sub SetTimeSpan(ByVal colID As Integer, ByVal val As TimeSpan)
```

C# syntax

```
public void SetTimeSpan(int colID, TimeSpan val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The set value is accurate to the millisecond and is normalized to a nonnegative value between 0 and 24 hours. The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.TimeSpan

SetToDefault method

Sets the value for the specified column to its default value.

Visual Basic syntax

```
Public Sub SetToDefault(ByVal colID As Integer)
```

C# syntax

```
public void SetToDefault(int colID)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "GetColumnDefaultValue method" on page 419
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252

SetUInt16 method

Sets the value for the specified column using a System.UInt16.

Visual Basic syntax

```
Public Sub SetUInt16(ByVal colID As Integer, ByVal val As UShort)
```

C# syntax

```
public void SetUInt16(int colID, ushort val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.UInt16

SetUInt32 method

Sets the value for the specified column using an System.UInt32.

Visual Basic syntax

```
Public Sub SetUInt32(ByVal colID As Integer, ByVal val As UInteger)
```

C# syntax

```
public void SetUInt32(int colID, uint val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.UInt32

SetUInt64 method

Sets the value for the specified column using a System.UInt64.

Visual Basic syntax

```
Public Sub SetUInt64(ByVal colID As Integer, ByVal val As ULong)
```

C# syntax

```
public void SetUInt64(int colID, ulong val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute an ULTable.Insert or Update, and that change is not made permanent until it is committed.

See also

- "GetOrdinal method" on page 239
- "Schema property" on page 255
- "GetFieldType method" on page 235
- "Insert method" on page 411
- "Update method" on page 357
- "FieldCount property" on page 252
- System.UInt64

Update method

Updates the current row with the current column values (specified using the set methods).

Visual Basic syntax

```
Public Sub Update()
```

C# syntax

```
public void Update()
```

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "UpdateBegin method" on page 358

UpdateBegin method

Prepares to update the current row.

Visual Basic syntax

```
Public Sub UpdateBegin()
```

C# syntax

```
public void UpdateBegin()
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Column values are modified by calling the appropriate setType or AppendType method(s). The first append on a column clears the current column value prior to appending the new value.

The data in the row is not actually changed until you call Update(), and that change is not made 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.

See also

- "Update method" on page 357

ULResultSetSchema class

UL Ext: Represents the schema of an UltraLite result set.

Visual Basic syntax

```
Public NotInheritable Class ULResultSetSchema Inherits ULCursorSchema
```

C# syntax

```
public sealed class ULResultSetSchema : ULCursorSchema
```

Base classes

- "ULCursorSchema class" on page 189

Members

All members of ULResultSetSchema class, including all inherited members.

Name	Description
"GetColumnID method"	Returns the column ID of the named column.
"GetColumnName method"	Returns the name of the column identified by the specified column ID.
"GetColumnPrecision method"	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnScale method"	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnSize method"	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
"GetColumnSQLName method"	Returns the name of the column identified by the specified column ID.
"GetColumnULDbType method"	Returns the UltraLite.NET data type of the column identified by the specified column ID.
"GetSchemaTable method"	Returns a System.Data.DataTable that describes the column schema of the ULDataReader.
"ColumnCount property"	Returns the number of columns in the cursor.
"IsOpen property"	Checks whether the cursor schema is currently open.
"Name property"	Returns the name of the cursor.

Remarks

There is no constructor for this class. A ULResultSetSchema object is attached to a result set as its ULDataReader.Schema.

A result set schema is only valid while the data reader is open.

See also

- "ULCommand class" on page 64
- "ULDataReader class" on page 220
- "ULResultSetSchema class" on page 358
- "Schema property" on page 255
- "ULCursorSchema class" on page 189

Name property

Returns the name of the cursor.

Visual Basic syntax

```
Public ReadOnly Overrides Property Name As String
```

C# syntax

```
public override string Name {get;}
```

Remarks

The SQL statement that generated the ULResultSetSchema.

ULRowsCopiedEventArgs class

Represents the set of arguments passed to the ULRowsCopiedEventHandler.

Visual Basic syntax

```
Public NotInheritable Class ULRowsCopiedEventArgs
```

C# syntax

```
public sealed class ULRowsCopiedEventArgs
```

Members

All members of ULRowsCopiedEventArgs class, including all inherited members.

Name	Description
"ULRowsCopiedEventArgs constructor"	Creates a new instance of the ULRowsCopiedEventArgs object.
"Abort property"	Gets or sets a value that indicates whether the bulk-copy operation should be aborted.
"RowsCopied property"	Returns the number of rows copied during the current bulk-copy operation.

Remarks

The ULRowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

See also

- "ULRowsCopiedEventHandler delegate" on page 434

ULRowsCopiedEventArgs constructor

Creates a new instance of the ULRowsCopiedEventArgs object.

Visual Basic syntax

```
Public Sub New(ByVal rowsCopied As Long)
```

C# syntax

```
public ULRowsCopiedEventArgs(long rowsCopied)
```

Parameters

- **rowsCopied** An 64-bit integer value that indicates the number of rows copied during the current bulk-copy operation.

Remarks

The ULRowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

Abort property

Gets or sets a value that indicates whether the bulk-copy operation should be aborted.

Visual Basic syntax

```
Public Property Abort As Boolean
```

C# syntax

```
public bool Abort {get;set;}
```

Remarks

The ULRowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

RowsCopied property

Returns the number of rows copied during the current bulk-copy operation.

Visual Basic syntax

```
Public ReadOnly Property RowsCopied As Long
```

C# syntax

```
public long RowsCopied {get;}
```

Remarks

A long integer representing the number of rows copied.

The ULRowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

ULRowUpdatedEventArgs class

Provides data for the ULDataAdapter.RowUpdated event.

Visual Basic syntax

```
Public NotInheritable Class ULRowUpdatedEventArgs
    Inherits System.Data.Common.RowUpdatedEventArgs
```

C# syntax

```
public sealed class ULRowUpdatedEventArgs :
    System.Data.Common.RowUpdatedEventArgs
```

Base classes

- System.Data.Common.RowUpdatedEventArgs

Members

All members of ULRowUpdatedEventArgs class, including all inherited members.

Name	Description
"ULRowUpdatedEventArgs constructor"	Initializes a new instance of the ULRowUpdatedEventArgs class.
CopyToRows method (Inherited from System.Data.Common.RowUpdatedEventArgs)	Copies references to the modified rows into the provided array.
"Command property"	Returns the ULCommand executed when DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping) is called.
Errors property (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the System.Data.Common.RowUpdatedEventArgs.Command was executed.
"RecordsAffected property"	Returns the number of rows changed, inserted, or deleted by the execution of the SQL statement.
Row property (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets the System.Data.DataRow sent through an System.Data.Common.DbDataAdapter.Update(System.Data.DataSet).
RowCount property (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets the number of rows processed in a batch of updated records.

Name	Description
StatementType property (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status property (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets the System.Data.UpdateStatus of the System.Data.Common.RowUpdatedEventArgs.Command property.
TableMapping property (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets the System.Data.Common.DataTableMapping sent through an System.Data.Common.DbDataAdapter.Update(System.Data.DataSet).

See also

- "RowUpdated event" on page 205
- System.Data.Common.RowUpdatedEventArgs

ULRowUpdatedEventArgs constructor

Initializes a new instance of the ULRowUpdatedEventArgs class.

Visual Basic syntax

```
Public Sub New(
    ByVal row As DataRow,
    ByVal command As IDbCommand,
    ByVal statementType As StatementType,
    ByVal tableMapping As DataTableMapping
)
```

C# syntax

```
public ULRowUpdatedEventArgs(
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
)
```

Parameters

- **row** The System.Data.DataRow sent through an DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping).
- **command** The System.Data.IDbCommand executed when DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping) is called.
- **statementType** One of the System.Data.StatementType values that specifies the type of query executed.

- **tableMapping** The System.Data.Common.DataTableMapping sent through an DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping).

See also

- System.Data.DataRow

Command property

Returns the ULCommand executed when DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping) is called.

Visual Basic syntax

```
Public ReadOnly Shadows Property Command As ULCommand
```

C# syntax

```
public new ULCommand Command {get;}
```

Remarks

The ULCommand object executed by the update.

This is the strongly-typed version of System.Data.Common.RowUpdatedEventArgs.Command.

See also

- "ULCommand class" on page 64
- System.Data.Common.RowUpdatedEventArgs.Command

RecordsAffected property

Returns the number of rows changed, inserted, or deleted by the execution of the SQL statement.

Visual Basic syntax

```
Public ReadOnly Shadows Property RecordsAffected As Integer
```

C# syntax

```
public new int RecordsAffected {get;}
```

Remarks

For SELECT statements this value is -1.

The number of rows changed, inserted, or deleted; 0 if no rows were affected or the statement failed; and -1 for SELECT statements.

ULRowUpdatingEventArgs class

Provides data for the ULDataAdapter.RowUpdating event.

Visual Basic syntax

```
Public NotInheritable Class ULRowUpdatingEventArgs
    Inherits System.Data.Common.RowUpdatingEventArgs
```

C# syntax

```
public sealed class ULRowUpdatingEventArgs :
    System.Data.Common.RowUpdatingEventArgs
```

Base classes

- System.Data.Common.RowUpdatingEventArgs

Members

All members of ULRowUpdatingEventArgs class, including all inherited members.

Name	Description
"ULRowUpdatingEventArgs constructor"	Initializes a new instance of the ULRowUpdatingEventArgs class.
BaseCommand property (Inherited from System.Data.Common.RowUpdatingEventArgs)	Gets or sets the System.Data.IDbCommand object for an instance of this class.
"Command property"	Specifies the ULCommand to execute when performing the DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping).
Errors property (Inherited from System.Data.Common.RowUpdatingEventArgs)	Gets any errors generated by the .NET Framework data provider when the System.Data.Common.RowUpdatedEventArgs.Command executes.
Row property (Inherited from System.Data.Common.RowUpdatingEventArgs)	Gets the System.Data.DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType property (Inherited from System.Data.Common.RowUpdatingEventArgs)	Gets the type of SQL statement to execute.
Status property (Inherited from System.Data.Common.RowUpdatingEventArgs)	Gets or sets the System.Data.UpdateStatus of the System.Data.Common.RowUpdatedEventArgs.Command property.

Name	Description
TableMapping property (Inherited from System.Data.Common.RowUpdatingEventArgs)	Gets the System.Data.Common.DataTableMapping to send through the System.Data.Common.DbDataAdapter.Update(System.Data.DataSet).

See also

- "RowUpdating event" on page 206
- System.Data.Common.RowUpdatingEventArgs

ULRowUpdatingEventArgs constructor

Initializes a new instance of the ULRowUpdatingEventArgs class.

Visual Basic syntax

```
Public Sub New(
    ByVal row As DataRow,
    ByVal command As IDbCommand,
    ByVal statementType As StatementType,
    ByVal tableMapping As DataTableMapping
)
```

C# syntax

```
public ULRowUpdatingEventArgs(
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
)
```

Parameters

- **row** The System.Data.DataRow to update.
- **command** The System.Data.IDbCommand to execute during the update.
- **statementType** One of the System.Data.StatementType values that specifies the type of query executed.
- **tableMapping** The System.Data.Common.DataTableMapping sent through an DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping).

See also

- System.Data.DataRow

Command property

Specifies the ULCommand to execute when performing the DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping).

Visual Basic syntax

```
Public Shadows Property Command As ULCommand
```

C# syntax

```
public new ULCommand Command {get;set;}
```

Remarks

The ULCommand object to execute when updating.

This is the strongly-typed version of System.Data.Common.RowUpdatingEventArgs.Command.

See also

- "ULCommand class" on page 64
- System.Data.Common.RowUpdatingEventArgs.Command

ULServerSyncListener interface

UL Ext: The listener interface for receiving server synchronization messages.

Visual Basic syntax

```
Public Interface ULServerSyncListener
```

C# syntax

```
public interface ULServerSyncListener
```

Members

All members of ULServerSyncListener interface, including all inherited members.

Name	Description
"ServerSyncInvoked method"	Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

ServerSyncInvoked method

Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

Visual Basic syntax

```
Public Sub ServerSyncInvoked(ByVal messageName As String)
```

C# syntax

```
public void ServerSyncInvoked(string messageName)
```

Parameters

- **messageName** The name of the message sent to the application.

Remarks

This method is invoked by a separate thread. To avoid multi-threading issues, it should post an event to the UI. If you are using multi-threading, it is recommended that you use a separate connection and use the lock keyword to access any objects shared with the rest of the application.

Example

The following Visual Basic code demonstrates how to receive a server synchronization request and perform a synchronization in the UI thread.

```
Imports iAnywhere.Data.UltraLite

Public Class MainWindow
    Inherits System.Windows.Forms.Form
    Implements ULServerSyncListener

    Private conn As ULConnection

    Public Sub New(ByVal args() As String)
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call
        ULConnection.DatabaseManager.SetServerSyncListener( _
            "myCompany.mymsg", "myCompany.myapp", Me _
        )
        'Create Connection
        ...
    End Sub

    Protected Overrides Sub OnClosing( _
        ByVal e As System.ComponentModel.CancelEventArgs _
    )
        ULConnection.DatabaseManager.SetServerSyncListener( _
            Nothing, Nothing, Nothing _
        )
        MyBase.OnClosing(e)
    End Sub

    Public Sub ServerSyncInvoked(ByVal messageName As String) _
        Implements ULServerSyncListener.ServerSyncInvoked

        Me.Invoke(New EventHandler(AddressOf Me.ServerSyncAction))
    End Sub

    Public Sub ServerSyncAction( _
        ByVal sender As Object, ByVal e As EventArgs _
```



```
        )
            ' Do Server sync
            conn.Synchronize()
        End Sub
    End Class
```

The following C# code demonstrates how to receive a server synchronization request and perform a synchronization in the UI thread.

```
using iAnywhere.Data.UltraLite;
public class Form1 : System.Windows.Forms.Form, ULServerSyncListener
{
    private System.Windows.Forms.MainMenu mainMenu1;
    private ULConnection conn;

    public Form1()
    {
        //
        // Required for Windows Form Designer support
        //
        InitializeComponent();

        //
        // TODO: Add any constructor code after
        // InitializeComponent call
        //
        ULConnection.DatabaseManager.SetServerSyncListener(
            "myCompany.mymsg", "myCompany.myapp", this
        );

        // Create connection
        ...
    }

    protected override void Dispose( bool disposing )
    {
        base.Dispose( disposing );
    }

    protected override void OnClosing(
        System.ComponentModel.CancelEventArgs e)
    {
        ULConnection.DatabaseManager.SetServerSyncListener(
            null, null, null
        );
        base.OnClosing(e);
    }

    public void ServerSyncInvoked( string messageName )
    {
        this.Invoke( new EventHandler( ServerSyncHandler ) );
    }

    internal void ServerSyncHandler(object sender, EventArgs e)
    {
        conn.Synchronize();
    }
}
```

ULSqlProgressData class

UL Ext: Returns SQL passthrough script progress monitoring data.

Visual Basic syntax

```
Public Class ULSqlProgressData
```

C# syntax

```
public class ULSqlProgressData
```

Members

All members of ULSqlProgressData class, including all inherited members.

Name	Description
"CurrentScript property"	The index of the scripts executed so far.
"ScriptCount property"	Returns the number of scripts being executed.
"State property"	Returns the current progress state.

CurrentScript property

The index of the scripts executed so far.

Visual Basic syntax

```
Public ReadOnly Property CurrentScript As Long
```

C# syntax

```
public long CurrentScript {get;}
```

Remarks

The current index of the scripts being executed.

ScriptCount property

Returns the number of scripts being executed.

Visual Basic syntax

```
Public ReadOnly Property ScriptCount As Long
```

C# syntax

```
public long ScriptCount {get;}
```


Remarks

The number of scripts being executed.

State property

Returns the current progress state.

Visual Basic syntax

```
Public ReadOnly Property State As ULSqlProgressState
```

C# syntax

```
public ULSqlProgressState State {get;}
```

Remarks

One of the ULSqlProgressState values specifying the current SQL pass through callback state.

See also

- "ULSyncProgressState enumeration" on page 443

ULSyncParms class

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

Visual Basic syntax

```
Public NotInheritable Class ULSyncParms
```

C# syntax

```
public sealed class ULSyncParms
```

Members

All members of ULSyncParms class, including all inherited members.

Name	Description
"CopyFrom method"	Copies the properties of the specified ULSyncParms object to this ULSyncParms object.
"ToString method"	Returns the string representation of this instance.
"AdditionalParms property"	Specifies additional synchronization parameters.
"AuthenticationParms property"	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Name	Description
“DownloadOnly property”	Specifies whether to disable or enable uploads when synchronizing.
“KeepPartialDownload property”	Specifies whether to disable or enable partial downloads when synchronizing.
“NewPassword property”	Specifies a new MobiLink password for the user specified with UserName.
“Password property”	The MobiLink password for the user specified by UserName.
“PingOnly property”	Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.
“Publications property”	Specifies the publications to be synchronized.
“ResumePartialDownload property”	Specifies whether to resume or discard a previous partial download.
“SendColumnNames property”	Specifies whether the client should send column names to the MobiLink server during synchronization.
“SendDownloadAck property”	Specifies whether the client should send a download acknowledgement to the MobiLink server during synchronization.
“Stream property”	Specifies the MobiLink synchronization stream to use for synchronization.
“StreamParms property”	Specifies the parameters to configure the synchronization stream.
“UploadOnly property”	Specifies whether to disable or enable downloads when synchronizing.
“UserName property”	The user name that uniquely identifies the MobiLink client to the MobiLink server.
“Version property”	Specifies which synchronization script to use.

Remarks

There is no constructor for this class. Each connection has its own ULSyncParms instance, attached as its ULConnection.SyncParms.

At most, only one synchronization command (ULSyncParms.DownloadOnly, ULSyncParms.PingOnly, ULSyncParms.ResumePartialDownload, or ULSyncParms.UploadOnly) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode.SQLE_SYNC_INFO_INVALID SQLException is thrown by ULConnection.Synchronize().

Other sources of ULSQLCode.SQLE_SYNC_INFO_INVALID errors include not specifying a ULSyncParms.Stream value or a ULSyncParms.Version value.

See also

- "ULConnection class" on page 110
- "SyncParms property" on page 151
- "Synchronize method" on page 141
- "DownloadOnly property" on page 374
- "PingOnly property" on page 377
- "ResumePartialDownload property" on page 378
- "UploadOnly property" on page 381
- "Synchronize method" on page 141
- "Stream property" on page 380
- "Version property" on page 382

CopyFrom method

Copies the properties of the specified ULSyncParms object to this ULSyncParms object.

Visual Basic syntax

```
Public Sub CopyFrom(ByVal src As ULSyncParms)
```

C# syntax

```
public void CopyFrom(ULSyncParms src)
```

Parameters

- **src** The object to copy from.

See also

- "ULSyncParms class" on page 371

ToString method

Returns the string representation of this instance.

Visual Basic syntax

```
Public Overrides Function ToString() As String
```

C# syntax

```
public override string ToString()
```

Returns

The string representation of this instance as a semicolon-separated list keyword=value pairs.

AdditionalParms property

Specifies additional synchronization parameters.

Visual Basic syntax

```
Public Property AdditionalParms As String
```

C# syntax

```
public string AdditionalParms {get;set;}
```

Remarks

A string, in the form of a semicolon-separated list of name=value pairs.

A number of additional synchronization parameters can be set with this property.

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Visual Basic syntax

```
Public Property AuthenticationParms As String()
```

C# syntax

```
public string[] AuthenticationParms {get;set;}
```

Remarks

An array of strings, each containing an authentication parameter (null array entries result in a synchronization error). The default is a null reference (Nothing in Visual Basic), meaning no authentication parameters.

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).

DownloadOnly property

Specifies whether to disable or enable uploads when synchronizing.

Visual Basic syntax

```
Public Property DownloadOnly As Boolean
```

C# syntax

```
public bool DownloadOnly {get;set;}
```


Remarks

True to disable uploads when synchronizing, false to enable uploads. The default is false.

At most, only one synchronization command (ULSyncParms.DownloadOnly, ULSyncParms.PingOnly, ULSyncParms.ResumePartialDownload, or ULSyncParms.UploadOnly) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode.SQLE_SYNC_INFO_INVALID SQLException is thrown by ULConnection.Synchronize().

See also

- "UploadOnly property" on page 381
- "DownloadOnly property" on page 374
- "PingOnly property" on page 377
- "ResumePartialDownload property" on page 378
- "UploadOnly property" on page 381
- "Synchronize method" on page 141

KeepPartialDownload property

Specifies whether to disable or enable partial downloads when synchronizing.

Visual Basic syntax

```
Public Property KeepPartialDownload As Boolean
```

C# syntax

```
public bool KeepPartialDownload {get;set;}
```

Remarks

True to enable partial downloads when synchronizing, false to disable partial downloads. The default is false.

UltraLite.NET has the ability to restart downloads that fail because of communication errors or user aborts through the ULSyncProgressListener. UltraLite.NET 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.NET should save partial downloads, specify connection.SyncParms.KeepPartialDownload=true; otherwise the download is rolled back if an error occurs.

If a partial download was kept, then the output field connection.SyncResult.ULSyncResult.PartialDownloadRetained is set to true when connection.Synchronize() exits.

If PartialDownloadRetained is set, then you can resume a download. To do this, call connection.Synchronize() with connection.SyncParms.ULSyncParms.ResumePartialDownload set to true. It is recommended that you keep KeepPartialDownload set to true as well in case another communications error occurs. 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 ULSyncParms fields are not relevant. For example, the Publications field is not used. You receive the publications that you requested on the initial download. The only fields that need to be set are ResumePartialDownload and UserName. The field KeepPartialDownload can be set if desired and function as normal.

If you have a partial download and it is no longer needed, then you can call ULConnection.RollbackPartialDownload() to roll back the failed download transaction. Also, if you attempt to synchronize again and do not specify ResumePartialDownload, then the partial download is rolled back before the next synchronization begins.

For more information, see the "Resuming failed downloads" [*MobiLink - Server Administration*].

See also

- "PartialDownloadRetained property" on page 392
- "ResumePartialDownload property" on page 378
- "RollbackPartialDownload method" on page 138
- "ResumePartialDownload property" on page 378
- "UserName property" on page 381
- "RollbackPartialDownload method" on page 138

NewPassword property

Specifies a new MobiLink password for the user specified with UserName.

Visual Basic syntax

```
Public Property NewPassword As String
```

C# syntax

```
public string NewPassword {get;set;}
```

Remarks

A string specifying a new MobiLink password. The default is a null reference (Nothing in Visual Basic), meaning the password is not changed.

A new password takes effect after the next synchronization.

See also

- "UserName property" on page 381

Password property

The MobiLink password for the user specified by UserName.

Visual Basic syntax

```
Public Property Password As String
```

C# syntax

```
public string Password {get;set;}
```

Remarks

A string specifying the MobiLink password. The default is a null reference (Nothing in Visual Basic), meaning no password is specified.

The MobiLink user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink server.

See also

- "NewPassword property" on page 376
- "UserName property" on page 381

PingOnly property

Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.

Visual Basic syntax

```
Public Property PingOnly As Boolean
```

C# syntax

```
public bool PingOnly {get;set;}
```

Remarks

True to specify that the client should only ping the MobiLink server, false to specify the client should perform a real synchronization. The default is false.

At most, only one synchronization command (ULSyncParms.DownloadOnly, ULSyncParms.PingOnly, ULSyncParms.ResumePartialDownload, or ULSyncParms.UploadOnly) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode.SQLE_SYNC_INFO_INVALID SQLException is thrown by ULConnection.Synchronize().

See also

- "DownloadOnly property" on page 374
- "PingOnly property" on page 377
- "ResumePartialDownload property" on page 378
- "UploadOnly property" on page 381
- "Synchronize method" on page 141

Publications property

Specifies the publications to be synchronized.

Visual Basic syntax

```
Public Property Publications As String
```

C# syntax

```
public string Publications {get;set;}
```

Remarks

A string containing a list of publication names, separated by comma (,); or the special value ULConnection.SYNC_ALL_PUBS, or the special value ULConnection.SYNC_ALL_DB. The default is ULConnection.SYNC_ALL_DB.

See also

- "SYNC_ALL_PUBS field" on page 154
- "SYNC_ALL_DB field" on page 154

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Visual Basic syntax

```
Public Property ResumePartialDownload As Boolean
```

C# syntax

```
public bool ResumePartialDownload {get;set;}
```

Remarks

True to resume a previous partial download, false to discard a previous partial download. The default is false.

Only at most one synchronization command (ULSyncParms.DownloadOnly, ULSyncParms.PingOnly, ULSyncParms.ResumePartialDownload, or ULSyncParms.UploadOnly) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode.SQLE_SYNC_INFO_INVALID SQLException is thrown by ULConnection.Synchronize().

See also

- "KeepPartialDownload property" on page 375
- "DownloadOnly property" on page 374
- "PingOnly property" on page 377
- "ResumePartialDownload property" on page 378
- "UploadOnly property" on page 381
- "Synchronize method" on page 141
- "PartialDownloadRetained property" on page 392

SendColumnNames property

Specifies whether the client should send column names to the MobiLink server during synchronization.

Visual Basic syntax

```
Public Property SendColumnNames As Boolean
```

C# syntax

```
public bool SendColumnNames {get;set;}
```

Remarks

True to specify that the client should send column names to the MobiLink server, false to specify that column names are not sent. The default is false.

The column names are used by the MobiLink server for direct row handling. When the MobiLink server is using the row handling API to refer to columns by name rather than by index, you should set this option. This is the only use of the column names that are sent by this option.

SendDownloadAck property

Specifies whether the client should send a download acknowledgement to the MobiLink server during synchronization.

Visual Basic syntax

```
Public Property SendDownloadAck As Boolean
```

C# syntax

```
public bool SendDownloadAck {get;set;}
```

Remarks

The download acknowledgement is sent after the download has been fully applied and committed at the remote (a positive acknowledgement) or after the download fails (a negative acknowledgement).

Set True to specify that the client should send a download acknowledgement to the MobiLink server. Set False to specify that no download acknowledgement is sent. The default is False.

If the client sends a download acknowledgement, the MobiLink server database worker thread must wait for the client to apply and commit the download. If the client does not sent a download acknowledgement, the MobiLink server is freed up sooner for its next synchronization.

Stream property

Specifies the MobiLink synchronization stream to use for synchronization.

Visual Basic syntax

```
Public Property Stream As ULStreamType
```

C# syntax

```
public ULStreamType Stream {get;set;}
```

Remarks

One of the ULStreamType values specifying the type of synchronization stream to use. The default is ULStreamType.TCPIP.

Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. These parameters are supplied by ULSyncParms.StreamParms.

If the stream type is set to a value that is invalid for the platform, the stream type is set to ULStreamType.TCPIP.

See also

- "ULStreamType enumeration" on page 443
- "StreamParms property" on page 380

StreamParms property

Specifies the parameters to configure the synchronization stream.

Visual Basic syntax

```
Public Property StreamParms As String
```

C# syntax

```
public string StreamParms {get;set;}
```

Remarks

A string, in the form of a semicolon-separated list of keyword-value pairs, specifying the parameters for the stream. The default is a null reference. (Nothing in Visual Basic)

For information about configuring specific stream types, see

"Network protocol options for UltraLite synchronization streams" [*UltraLite - Database Management and Reference*].

StreamParms is 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").

See also

- "Stream property" on page 380
- "ULStreamType enumeration" on page 443

UploadOnly property

Specifies whether to disable or enable downloads when synchronizing.

Visual Basic syntax

```
Public Property UploadOnly As Boolean
```

C# syntax

```
public bool UploadOnly {get;set;}
```

Remarks

True to disable downloads, false to enable downloads. The default is false.

At most, only one synchronization command (ULSyncParms.DownloadOnly, ULSyncParms.PingOnly, ULSyncParms.ResumePartialDownload, or ULSyncParms.UploadOnly) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode.SQLE_SYNC_INFO_INVALID SQLException is thrown by ULConnection.Synchronize().

See also

- "DownloadOnly property" on page 374
- "PingOnly property" on page 377
- "ResumePartialDownload property" on page 378
- "UploadOnly property" on page 381
- "Synchronize method" on page 141

UserName property

The user name that uniquely identifies the MobiLink client to the MobiLink server.

Visual Basic syntax

```
Public Property UserName As String
```

C# syntax

```
public string UserName {get;set;}
```

Remarks

A string specifying the user name. This parameter has no default value, and must be explicitly set.

The MobiLink server 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 server.

See also

- "Password property" on page 376

Version property

Specifies which synchronization script to use.

Visual Basic syntax

```
Public Property Version As String
```

C# syntax

```
public string Version {get;set;}
```

Remarks

A string specifying the version of the synchronization script to use. This parameter has no default value, and must be explicitly set.

Each synchronization script in the consolidated database is marked with a version string. For example, there can be two different download_cursor scripts, with each one identified by a different version string. The version string allows an UltraLite application to choose from a set of synchronization scripts.

ULSyncProgressData class

UL Ext: Returns synchronization progress monitoring data.

Visual Basic syntax

```
Public Class ULSyncProgressData
```

C# syntax

```
public class ULSyncProgressData
```

Members

All members of ULSyncProgressData class, including all inherited members.

Name	Description
“Flags property”	Returns the current synchronization flags indicating additional information relating to the current state.
“ReceivedBytes property”	Returns the number of bytes received so far.
“ReceivedDeletes property”	Returns the number of deleted rows received so far.
“ReceivedInserts property”	Returns the number of inserted rows received so far.
“ReceivedUpdates property”	Returns the number of updated rows received so far.
“SentBytes property”	Returns the number of bytes sent so far.
“SentDeletes property”	Returns the number of deleted rows sent so far.
“SentInserts property”	Returns the number of inserted rows sent so far.
“SentUpdates property”	Returns the number of updated rows sent so far.
“State property”	Returns the current synchronization state.
“SyncTableCount property”	Returns the number of tables being synchronized.
“SyncTableIndex property”	Returns the index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).
“TableID property”	Returns the database index of the table currently being synchronized.
“TableName property”	Returns the name of the current table being uploaded or downloaded.
“FLAG_IS_BLOCKING field”	A flag indicating that the synchronization is blocked awaiting a response from the MobiLink server.

See also

- “ULSyncProgressListener interface” on page 389

Flags property

Returns the current synchronization flags indicating additional information relating to the current state.

Visual Basic syntax

```
Public ReadOnly Property Flags As Integer
```

C# syntax

```
public int Flags {get;}
```

Remarks

An integer containing a combination of flags or'ed together.

See also

- "FLAG_IS_BLOCKING field" on page 389

ReceivedBytes property

Returns the number of bytes received so far.

Visual Basic syntax

```
Public ReadOnly Property ReceivedBytes As Long
```

C# syntax

```
public long ReceivedBytes {get;}
```

Remarks

This information is updated for all states.

The number of bytes received so far.

See also

- "ULSyncProgressState enumeration" on page 443

ReceivedDeletes property

Returns the number of deleted rows received so far.

Visual Basic syntax

```
Public ReadOnly Property ReceivedDeletes As Integer
```

C# syntax

```
public int ReceivedDeletes {get;}
```

Remarks

The number of deleted rows received so far.

See also

- "ULSyncProgressState enumeration" on page 443

ReceivedInserts property

Returns the number of inserted rows received so far.

Visual Basic syntax

```
Public ReadOnly Property ReceivedInserts As Integer
```

C# syntax

```
public int ReceivedInserts {get;}
```

Remarks

The number of inserted rows received so far.

See also

- "ULSyncProgressState enumeration" on page 443

ReceivedUpdates property

Returns the number of updated rows received so far.

Visual Basic syntax

```
Public ReadOnly Property ReceivedUpdates As Integer
```

C# syntax

```
public int ReceivedUpdates {get;}
```

Remarks

The number of updated rows received so far.

See also

- "ULSyncProgressState enumeration" on page 443

SentBytes property

Returns the number of bytes sent so far.

Visual Basic syntax

```
Public ReadOnly Property SentBytes As Long
```

C# syntax

```
public long SentBytes {get;}
```

Remarks

This information is updated for all states.

The number of bytes sent so far.

See also

- "ULSyncProgressState enumeration" on page 443

SentDeletes property

Returns the number of deleted rows sent so far.

Visual Basic syntax

```
Public ReadOnly Property SentDeletes As Integer
```

C# syntax

```
public int SentDeletes {get;}
```

Remarks

The number of deleted rows sent so far.

See also

- "ULSyncProgressState enumeration" on page 443

SentInserts property

Returns the number of inserted rows sent so far.

Visual Basic syntax

```
Public ReadOnly Property SentInserts As Integer
```

C# syntax

```
public int SentInserts {get;}
```

Remarks

The number of inserted rows sent so far.

See also

- "ULSyncProgressState enumeration" on page 443

SentUpdates property

Returns the number of updated rows sent so far.

Visual Basic syntax

```
Public ReadOnly Property SentUpdates As Integer
```

C# syntax

```
public int SentUpdates {get;}
```

Remarks

The number of updated rows sent so far.

See also

- "ULSyncProgressState enumeration" on page 443

State property

Returns the current synchronization state.

Visual Basic syntax

```
Public ReadOnly Property State As ULSyncProgressState
```

C# syntax

```
public ULSyncProgressState State {get;}
```

Remarks

One of the ULSyncProgressState values specifying the current synchronization state.

See also

- "ULSyncProgressState enumeration" on page 443

SyncTableCount property

Returns the number of tables being synchronized.

Visual Basic syntax

```
Public ReadOnly Property SyncTableCount As Integer
```

C# syntax

```
public int SyncTableCount {get;}
```

Remarks

The number of tables being synchronized. For each table there is a sending and receiving phase, so this number may be more than the number of tables being synchronized.

See also

- "ULSyncProgressState enumeration" on page 443

SyncTableIndex property

Returns the index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).

Visual Basic syntax

```
Public ReadOnly Property SyncTableIndex As Integer
```

C# syntax

```
public int SyncTableIndex {get;}
```

Remarks

The index of the table currently being synchronized, in the range from 1 to SyncTableCount

See also

- "ULSyncProgressState enumeration" on page 443

TableID property

Returns the database index of the table currently being synchronized.

Visual Basic syntax

```
Public ReadOnly Property TableID As Integer
```

C# syntax

```
public int TableID {get;}
```

Remarks

The database index, in the range from 1 to ULDatabaseSchema.TableCount

See also

- "ULSyncProgressState enumeration" on page 443
- "TableCount property" on page 220

TableName property

Returns the name of the current table being uploaded or downloaded.

Visual Basic syntax

```
Public ReadOnly Property TableName As String
```

C# syntax

```
public string TableName {get;}
```

Remarks

Name of the current table being synchronized; null if not applicable.

FLAG_IS_BLOCKING field

A flag indicating that the synchronization is blocked awaiting a response from the MobiLink server.

Visual Basic syntax

```
Public Const FLAG_IS_BLOCKING As Integer
```

C# syntax

```
public const int FLAG_IS_BLOCKING;
```

ULSyncProgressListener interface

UL Ext: The listener interface for receiving synchronization progress events.

Visual Basic syntax

```
Public Interface ULSyncProgressListener
```

C# syntax

```
public interface ULSyncProgressListener
```

Members

All members of ULSyncProgressListener interface, including all inherited members.

Name	Description
"SyncProgressed method"	Invoked during synchronization to inform the user of progress.

See also

- "Synchronize method" on page 141

SyncProgressed method

Invoked during synchronization to inform the user of progress.

Visual Basic syntax

```
Public Function SyncProgressed(
    ByVal data As ULSyncProgressData
) As Boolean
```

C# syntax

```
public bool SyncProgressed(ULSyncProgressData data)
```

Parameters

- **data** A ULSyncProgressData object containing the latest synchronization progress data.

Returns

This method should return true to cancel synchronization or return false to continue.

Remarks

This method should return true to cancel synchronization or return false to continue.

No UltraLite.NET API methods should be invoked during a SyncProgressed call.

See also

- "ULSyncProgressData class" on page 382

ULSyncResult class

UL Ext: Represents the status of the last synchronization.

Visual Basic syntax

```
Public Class ULSyncResult
```

C# syntax

```
public class ULSyncResult
```

Members

All members of ULSyncResult class, including all inherited members.

Name	Description
"AuthStatus property"	Returns the authorization status code for the last synchronization attempt.
"AuthValue property"	Returns the return value from custom user authentication synchronization scripts.

Name	Description
"IgnoredRows property"	Checks whether any uploaded rows were ignored during the last synchronization.
"PartialDownloadRetained property"	Checks whether a partial download was retained during the last synchronization.
"StreamErrorCode property"	Returns the error reported by the stream itself.
"StreamErrorParameters property"	Returns a comma-separated list of stream error parameters.
"StreamErrorSystem property"	Returns the stream error system-specific code.
"Timestamp property"	Returns the timestamp of the last synchronization.
"UploadOK property"	Checks whether the last upload synchronization was successful.

Remarks

There is no constructor for this class. Each connection has its own ULSyncResult instance, attached as its ULConnection.SyncResult. A ULSyncResult instance is only valid while that connection is open.

See also

- "SyncResult property" on page 151
- "Synchronize method" on page 141

AuthStatus property

Returns the authorization status code for the last synchronization attempt.

Visual Basic syntax

```
Public ReadOnly Property AuthStatus As ULAuthStatusCode
```

C# syntax

```
public ULAuthStatusCode AuthStatus {get;}
```

Remarks

One of the ULAuthStatusCode values denoting the authorization status for the last synchronization attempt.

See also

- "ULAuthStatusCode enumeration" on page 435

AuthValue property

Returns the return value from custom user authentication synchronization scripts.

Visual Basic syntax

```
Public ReadOnly Property AuthValue As Long
```

C# syntax

```
public long AuthValue {get;}
```

Remarks

A long integer returned from custom user authentication synchronization scripts.

IgnoredRows property

Checks whether any uploaded rows were ignored during the last synchronization.

Visual Basic syntax

```
Public ReadOnly Property IgnoredRows As Boolean
```

C# syntax

```
public bool IgnoredRows {get;}
```

Remarks

True if any uploaded rows were ignored during the last synchronization, false if no rows were ignored.

See also

- "DownloadOnly property" on page 374

PartialDownloadRetained property

Checks whether a partial download was retained during the last synchronization.

Visual Basic syntax

```
Public ReadOnly Property PartialDownloadRetained As Boolean
```

C# syntax

```
public bool PartialDownloadRetained {get;}
```

Remarks

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.

See also

- "KeepPartialDownload property" on page 375

StreamErrorCode property

Returns the error reported by the stream itself.

Visual Basic syntax

```
Public ReadOnly Property StreamErrorCode As ULStreamErrorCode
```

C# syntax

```
public ULStreamErrorCode StreamErrorCode {get;}
```

Remarks

One of the ULStreamErrorCode values denoting the error reported by the stream itself, ULStreamErrorCode.NONE if no error occurred.

StreamErrorParameters property

Returns a comma-separated list of stream error parameters.

Visual Basic syntax

```
Public ReadOnly Property StreamErrorParameters As String
```

C# syntax

```
public string StreamErrorParameters {get;}
```

Remarks

Contains a comma separated list of error parameters for the stream error code reported in StreamErrorCode property. This is an empty string either for errors with no parameters, or when no error has been set.

See also

- "StreamErrorCode property" on page 277

StreamErrorSystem property

Returns the stream error system-specific code.

Visual Basic syntax

```
Public ReadOnly Property StreamErrorSystem As Integer
```

C# syntax

```
public int StreamErrorSystem {get;}
```

Remarks

An integer denoting the stream error system-specific code.

Timestamp property

Returns the timestamp of the last synchronization.

Visual Basic syntax

```
Public ReadOnly Property Timestamp As Date
```

C# syntax

```
public DateTime Timestamp {get;}
```

Remarks

A System.DateTime specifying the timestamp of the last synchronization.

See also

- System.DateTime

UploadOK property

Checks whether the last upload synchronization was successful.

Visual Basic syntax

```
Public ReadOnly Property UploadOK As Boolean
```

C# syntax

```
public bool UploadOK {get;}
```

Remarks

True if the last upload synchronization was successful, false if the last upload synchronization was unsuccessful.

ULTable class

UL Ext: Represents a table in an UltraLite database.

Visual Basic syntax

```
Public Class ULTable Inherits ULResultSet
```

C# syntax

```
public class ULTable : ULResultSet
```

Base classes

- "ULResultSet class" on page 334

Members

All members of ULTable class, including all inherited members.

Name	Description
"AppendBytes method"	Appends the specified subset of the specified array of System.Bytes to the new value for the specified ULDbType.LongBinary column.
"AppendChars method"	Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType.LongVarchar column.
"Close method"	Closes the cursor.
"Delete method"	Deletes the current row.
"DeleteAllRows method"	Deletes all rows in the table.
Dispose method (Inherited from System.Data.Common.DbDataReader)	Releases all resources used by the current instance of the System.Data.Common.DbDataReader class.
"FindBegin method"	Prepares to perform a new Find on a 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.

Name	Description
“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.
“FindNext method”	Continues a ULTable.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.
“FindPrevious method”	Continues a ULTable.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.
“GetBoolean method”	Returns the value for the specified column as a System.Boolean.
“GetByte method”	Returns the value for the specified column as an unsigned 8-bit value (System.Byte).
“GetBytes method”	**UL Ext:** Returns the value for the specified column as an array of System.Bytes.
“GetChar method”	This method is not supported in UltraLite.NET.
“GetChars method”	Copies a subset of the value for the specified ULDbType.LongVarchar column, beginning at the specified offset, to the specified offset of the destination System.Char array.

Name	Description
GetData method (Inherited from System.Data.Common.DbDataReader)	Returns a System.Data.Common.DbDataReader object for the requested column ordinal.
"GetDataTypeName method"	Returns the name of the specified column's provider data type.
"GetDateTime method"	Returns the value for the specified column as a System.DateTime with millisecond accuracy.
GetDbDataReader method (Inherited from System.Data.Common.DbDataReader)	Returns a System.Data.Common.DbDataReader object for the requested column ordinal that can be overridden with a provider-specific implementation.
"GetDecimal method"	Returns the value for the specified column as a System.Decimal.
"GetDouble method"	Returns the value for the specified column as a System.Double.
"GetEnumerator method"	Returns an System.Collections.IEnumerator that iterates through the ULDataReader.
"GetFieldType method"	Returns the System.Type most appropriate for the specified column.
"GetFloat method"	Returns the value for the specified column as a System.Single.
"GetGuid method"	Returns the value for the specified column as a UUID (System.Guid).
"GetInt16 method"	Returns the value for the specified column as a System.Int16.
"GetInt32 method"	Returns the value for the specified column as an Int32.

Name	Description
"GetInt64 method"	Returns the value for the specified column as an Int64.
"GetName method"	Returns the name of the specified column.
"GetOrdinal method"	Returns the column ID of the named column.
GetProviderSpecificFieldType method (Inherited from System.Data.Common.DbDataReader)	Returns the provider-specific field type of the specified column.
GetProviderSpecificValue method (Inherited from System.Data.Common.DbDataReader)	Gets the value of the specified column as an instance of System.Object.
GetProviderSpecificValues method (Inherited from System.Data.Common.DbDataReader)	Gets all provider-specific attribute columns in the collection for the current row.
"GetRowCount method"	**UL Ext:** Returns the number of rows in the cursor, within threshold.
"GetSchemaTable method"	Returns a System.Data.DataTable that describes the column metadata of the ULDataReader.
"GetString method"	Returns the value for the specified column as a System.String.
"GetTimeSpan method"	Returns the value for the specified column as a System.TimeSpan with millisecond accuracy.
"GetUInt16 method"	Returns the value for the specified column as a System.UInt16.
"GetUInt32 method"	Returns the value for the specified column as a UInt32.
"GetUInt64 method"	Returns the value for the specified column as a System.UInt64.

Name	Description
“GetValue method”	Returns the value of the specified column in its native format.
“GetValues method”	Returns all the column values for the current row.
“Insert method”	Inserts a new row with the current column values (specified using the set methods).
“InsertBegin method”	Prepares to insert a new row into the table by setting all current column values to their default values.
“IsDBNull method”	Checks whether the value from the specified column is NULL.
“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.
“LookupBegin method”	Prepares to perform a new lookup on the table.
“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.
“MoveAfterLast method”	**UL Ext:** Positions the cursor to after the last row of the cursor.
“MoveBeforeFirst method”	**UL Ext:** Positions the cursor to before the first row of the cursor.
“MoveFirst method”	**UL Ext:** Positions the cursor to the first row of the cursor.
“MoveLast method”	**UL Ext:** Positions the cursor to the last row of the cursor.

Name	Description
“MoveNext method”	**UL Ext:** Positions the cursor to the next row or after the last row if the cursor was already on the last row.
“MovePrevious method”	**UL Ext:** Positions the cursor to the previous row or before the first row.
“MoveRelative method”	**UL Ext:** Positions the cursor relative to the current row.
“NextResult method”	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
“Read method”	Positions the cursor to the next row, or after the last row if the cursor was already on the last row.
“SetBoolean method”	Sets the value for the specified column using a System.Boolean.
“SetByte method”	Sets the value for the specified column using a System.Byte (unsigned 8-bit integer).
“SetBytes method”	Sets the value for the specified column using an array of System.Bytes.
“SetDateTime method”	Sets the value for the specified column using a System.DateTime.
“SetDBNull method”	Sets a column to NULL.
“SetDecimal method”	Sets the value for the specified column using a System.Decimal.
“SetDouble method”	Sets the value for the specified column using a System.Double.
“SetFloat method”	Sets the value for the specified column using a System.Single.

Name	Description
“SetGuid method”	Sets the value for the specified column using a System.Guid.
“SetInt16 method”	Sets the value for the specified column using an System.Int16.
“SetInt32 method”	Sets the value for the specified column using an System.Int32.
“SetInt64 method”	Sets the value for the specified column using an Int64.
“SetString method”	Sets the value for the specified column using a System.String.
“SetTimeSpan method”	Sets the value for the specified column using a System.Time-Span.
“SetToDefault method”	Sets the value for the specified column to its default value.
“SetUInt16 method”	Sets the value for the specified column using a System.UInt16.
“SetUInt32 method”	Sets the value for the specified column using an System.UInt32.
“SetUInt64 method”	Sets the value for the specified column using a System.UInt64.
“Truncate method”	Deletes all rows in the table while temporarily activating a stop synchronization delete.
“Update method”	Updates the current row with the current column values (specified using the set methods).
“UpdateBegin method”	Prepares to update the current row.
“Depth property”	Returns the depth of nesting for the current row.
“FieldCount property”	Returns the number of columns in the cursor.

Name	Description
"HasRows property"	Checks whether the ULDataReader has one or more rows.
"IsBOF property"	**UL Ext:** Checks whether the current row position is before the first row.
"IsClosed property"	Checks whether the cursor is currently open.
"IsEOF property"	**UL Ext:** Checks whether the current row position is after the last row.
"RecordsAffected property"	Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.
"RowCount property"	**UL Ext:** Returns the number of rows in the cursor.
"Schema property"	Holds the table schema.
"this property"	Returns the value of the specified column in its native format.
VisibleFieldCount property (Inherited from System.Data.Common.DbDataReader)	Gets the number of fields in the System.Data.Common.DbDataReader that are not hidden.

Remarks

There is no constructor for this class. Tables are created using the ULCommand.ExecuteTable() of the ULCommand.

See also

- "ExecuteTable method" on page 91
- "ULCommand class" on page 64
- "ULResultSet class" on page 334
- System.Data.IDataReader

DeleteAllRows method

Deletes all rows in the table.

Visual Basic syntax

```
Public Sub DeleteAllRows()
```

C# syntax

```
public void DeleteAllRows()
```

Exceptions

- **"ULException class"** A SQL error occurred.

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 ULConnection.StopSynchronizationDelete.

See also

- "Truncate method" on page 416
- "StopSynchronizationDelete method" on page 141

FindBegin method

Prepares to perform a new Find on a table.

Visual Basic syntax

```
Public Sub FindBegin()
```

C# syntax

```
public void FindBegin()
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The value(s) for which to search are specified by calling the appropriate setType method(s) on the columns in the index with which the table was opened.

See also

- "FindFirst method" on page 403
- "FindLast method" on page 405

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.

Overload list

Name	Description
"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.
"FindFirst(short) 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.

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.

Visual Basic syntax

```
Public Function FindFirst() As Boolean
```

C# syntax

```
public bool FindFirst()
```

Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure, the cursor position is after the last row (ULDataReader.IsEOF).

Each search must be preceded by a call to FindBegin.

See also

- "FindBegin method" on page 403
- "FindNext method" on page 407
- "FindPrevious method" on page 409
- "FindFirst method" on page 403
- "IsEOF property" on page 254
- "FindBegin method" on page 403

FindFirst(short) 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.

Visual Basic syntax

```
Public Function FindFirst(ByVal numColumns As Short) As Boolean
```

C# syntax

```
public bool FindFirst(short numColumns)
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the find. 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 value of 1.

Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure, the cursor position is after the last row (ULDataReader.IsEOF).

Each search must be preceded by a call to FindBegin.

See also

- "FindBegin method" on page 403
- "FindNext method" on page 407
- "FindPrevious method" on page 409
- "FindFirst method" on page 403
- "IsEOF property" on page 254
- "FindBegin method" on page 403

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.

Overload list

Name	Description
"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
"FindLast(short) 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.

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.

Visual Basic syntax

```
Public Function FindLast() As Boolean
```

C# syntax

```
public bool FindLast()
```

Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is before the first row (ULDataReader.IsBOF).

Each search must be preceded by a call to FindBegin.

See also

- "FindBegin method" on page 403
- "FindNext method" on page 407
- "FindPrevious method" on page 409
- "FindLast method" on page 405
- "IsBOF property" on page 253
- "FindBegin method" on page 403

FindLast(short) 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.

Visual Basic syntax

```
Public Function FindLast(ByVal numColumns As Short) As Boolean
```


C# syntax

```
public bool FindLast(short numColumns)
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the find. For example, if you have a three column index and you want to find a value that matches based on the first column only, you should set the value for the first column, then supply a value of 1.

Returns

True if successful, false otherwise

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is before the first row (ULDataReader.IsBOF).

Each search must be preceded by a call to FindBegin.

See also

- "FindBegin method" on page 403
- "FindNext method" on page 407
- "FindPrevious method" on page 409
- "FindLast method" on page 405
- "IsBOF property" on page 253
- "FindBegin method" on page 403

FindNext method

Continues a ULTable.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.

Overload list

Name	Description
"FindNext() method"	Continues a ULTable.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.
"FindNext(short) method"	Continues a ULTable.FindFirst() 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.

FindNext() method

Continues a ULTable.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.

Visual Basic syntax

```
Public Function FindNext() As Boolean
```

C# syntax

```
public bool FindNext()
```

Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

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 (ULDataReader.IsEOF).

FindNext behavior is undefined if the column values being searched for are modified during a row update.

See also

- "FindFirst method" on page 403
- "FindNext method" on page 407
- "IsEOF property" on page 254

FindNext(short) method

Continues a ULTable.FindFirst() 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.

Visual Basic syntax

```
Public Function FindNext(ByVal numColumns As Short) As Boolean
```

C# syntax

```
public bool FindNext(short numColumns)
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the find. For example, if you have a three column index, and you want to find a value that matches based on the first column only, you should set the value for the first column, and then supply a value of 1.

Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

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 (ULDataReader.IsEOF).

FindNext behavior is undefined if the column values being searched for are modified during a row update.

See also

- "FindFirst method" on page 403
- "FindNext method" on page 407
- "FindFirst method" on page 403
- "IsEOF property" on page 254

FindPrevious method

Continues a ULTable.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.

Overload list

Name	Description
"FindPrevious() method"	Continues a ULTable.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.
"FindPrevious(short) method"	Continues a ULTable.FindLast() 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.

FindPrevious() method

Continues a ULTable.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.

Visual Basic syntax

```
Public Function FindPrevious() As Boolean
```

C# syntax

```
public bool FindPrevious()
```

Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

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 (ULDataReader.IsBOF).

FindPrevious behavior is undefined if the column values being searched for are modified during a row update.

See also

- "FindLast method" on page 405
- "FindPrevious method" on page 409
- "IsBOF property" on page 253

FindPrevious(short) method

Continues a ULTable.FindLast() 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.

Visual Basic syntax

```
Public Function FindPrevious(ByVal numColumns As Short) As Boolean
```

C# syntax

```
public bool FindPrevious(short numColumns)
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the find. 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, then supply a value of 1.

Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

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 (ULDataReader.IsBOF).

FindPrevious behavior is undefined if the column values being searched for are modified during a row update.

See also

- "FindLast method" on page 405
- "FindPrevious method" on page 409
- "IsBOF property" on page 253

Insert method

Inserts a new row with the current column values (specified using the set methods).

Visual Basic syntax

```
Public Sub Insert()
```

C# syntax

```
public void Insert()
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Each insert must be preceded by a call to ULTable.InsertBegin.

See also

- "InsertBegin method" on page 411

InsertBegin method

Prepares to insert a new row into the table by setting all current column values to their default values.

Visual Basic syntax

```
Public Sub InsertBegin()
```

C# syntax

```
public void InsertBegin()
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Call the appropriate SetType or AppendType method(s) to specify the non-default values that are to be inserted.

The row is not actually inserted and the data in the row is not actually changed until you execute the Insert method, and that change is not made permanent until it is committed.

See also

- "Insert method" on page 411

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.

Overload list

Name	Description
"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.
"LookupBackward(short) 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.

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.

Visual Basic syntax

```
Public Function LookupBackward() As Boolean
```

C# syntax

```
public bool LookupBackward()
```

Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that 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 (ULDataReader.IsBOF).

Each search must be preceded by a call to LookupBegin.

See also

- "LookupBegin method" on page 413
- "LookupBackward method" on page 412
- "IsBOF property" on page 253

LookupBackward(short) 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.

Visual Basic syntax

```
Public Function LookupBackward(ByVal numColumns As Short) As Boolean
```

C# syntax

```
public bool LookupBackward(short 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 value of 1.

Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that 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 (ULDataReader.IsBOF).

Each search must be preceded by a call to LookupBegin.

See also

- "LookupBegin method" on page 413
- "LookupBackward method" on page 412
- "IsBOF property" on page 253

LookupBegin method

Prepares to perform a new lookup on the table.

Visual Basic syntax

```
Public Sub LookupBegin()
```

C# syntax

```
public void LookupBegin()
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The value(s) for which to search are specified by calling the appropriate setType method(s) on the columns in the index with which the table was opened.

See also

- "LookupForward method" on page 414
- "LookupBackward method" on page 412

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.

Overload list

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.
"LookupForward(short) 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.

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.

Visual Basic syntax

```
Public Function LookupForward() As Boolean
```

C# syntax

```
public bool LookupForward()
```


Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that 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 (ULDataReader.IsEOF).

Each search must be preceded by a call to ULTable.LookupBegin().

See also

- "LookupBegin method" on page 413
- "LookupForward method" on page 414
- "IsEOF property" on page 254

LookupForward(short) 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.

Visual Basic syntax

```
Public Function LookupForward(ByVal numColumns As Short) As Boolean
```

C# syntax

```
public bool LookupForward(short 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 value of 1.

Returns

True if successful, false otherwise.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that 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 ULDataReader.IsEOF).

Each search must be preceded by a call to LookupBegin.

See also

- "LookupBegin method" on page 413
- "LookupForward method" on page 414
- "IsEOF property" on page 254
- "LookupBegin method" on page 413

Truncate method

Deletes all rows in the table while temporarily activating a stop synchronization delete.

Visual Basic syntax

```
Public Sub Truncate()
```

C# syntax

```
public void Truncate()
```

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "DeleteAllRows method" on page 402

Schema property

Holds the table schema.

Visual Basic syntax

```
Public ReadOnly Shadows Property Schema As ULTableSchema
```

C# syntax

```
public new ULTableSchema Schema {get;}
```

Remarks

This property is only valid while its connection is open.

The ULTableSchema object representing the table schema.

This property represents the complete schema of the table, including UltraLite.NET extended information which is not represented in the results from ULDataReader.GetSchemaTable.

See also

- "ULTableSchema class" on page 417

ULTableSchema class

UL Ext: Represents the schema of an UltraLite table.

Visual Basic syntax

```
Public NotInheritable Class ULTableSchema Inherits ULCursorSchema
```

C# syntax

```
public sealed class ULTableSchema : ULCursorSchema
```

Base classes

- "ULCursorSchema class" on page 189

Members

All members of ULTableSchema class, including all inherited members.

Name	Description
"GetColumnDefaultValue method"	Returns the default value of the specified column.
"GetColumnID method"	Returns the column ID of the named column.
"GetColumnName method"	Returns the name of the column identified by the specified column ID.
"GetColumnPartitionSize method"	Returns the global autoincrement partition size assigned to the specified column.
"GetColumnPrecision method"	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnScale method"	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnSize method"	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
"GetColumnSQLName method"	Returns the name of the column identified by the specified column ID.

Name	Description
“GetColumnULDbType method”	Returns the UltraLite.NET data type of the column identified by the specified column ID.
“GetIndex method”	Returns the index schema of the named index.
“GetIndexName method”	Returns the name of the index identified by the specified index ID.
“GetOptimalIndex method”	The optimal index for searching a table using the specified column.
“GetPublicationPredicate method”	Returns the publication predicate for this table in the named publication.
“GetSchemaTable method”	Returns a System.Data.DataTable that describes the column schema of the ULDataReader.
“IsColumnAutoIncrement 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 (ULDbType.Date).
“IsColumnCurrentTime method”	Checks whether the specified column's default is set to the current time (ULDbType.Time).
“IsColumnCurrentTimestamp method”	Checks whether the specified column's default is set to the current timestamp (ULDbType.TimeStamp).
“IsColumnCurrentUTCTimestamp method”	Checks whether the specified column's default is set to the current UTC timestamp (ULDbType.TimeStamp).
“IsColumnGlobalAutoIncrement method”	Checks whether the specified column's default is set to global autoincrement.
“IsColumnNewUUID method”	Checks whether the specified column's default is set to a new UUID (System.Guid).
“IsColumnNullable method”	Checks whether the specified column is nullable.
“IsInPublication method”	Checks whether the table is contained in the named publication.
“ColumnCount property”	Returns the number of columns in the cursor.
“IndexCount property”	Returns the number of indexes on the table.

Name	Description
"IsNeverSynchronized property"	Checks whether the table is marked as never being synchronized.
"IsOpen property"	Checks whether the cursor schema is currently open.
"Name property"	Returns the name of the table.
"PrimaryKey property"	Returns the index schema of the primary key for the table.
"UploadUnchangedRows property"	Checks whether the database uploads rows that have not changed.

Remarks

There is no constructor for this class. A ULTableSchema object is attached to a table as its ULTable.Schema.

See also

- "ULTableSchema class" on page 417
- "Schema property" on page 416
- "ULCursorSchema class" on page 189

GetColumnDefaultValue method

Returns the default value of the specified column.

Visual Basic syntax

```
Public Function GetColumnDefaultValue(
    ByVal columnID As Integer
) As String
```

C# syntax

```
public string GetColumnDefaultValue(int columnID)
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in a table has an ID value of zero.

Returns

The default value of the specified column as a string or a null reference (Nothing in Visual Basic) if the default value is null.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ColumnCount property" on page 195

GetColumnPartitionSize method

Returns the global autoincrement partition size assigned to the specified column.

Visual Basic syntax

```
Public Function GetColumnPartitionSize(
    ByVal columnID As Integer
) As ULong
```

C# syntax

```
public ulong GetColumnPartitionSize(int columnID)
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

The column's global autoincrement partition size as a System.UInt64.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

All global autoincrement columns in a given table share the same global autoincrement partition.

See also

- "IsColumnGlobalAutoIncrement method" on page 425
- "ColumnCount property" on page 195
- System.UInt64

GetIndex method

Returns the index schema of the named index.

Visual Basic syntax

```
Public Function GetIndex(ByVal name As String) As ULIndexSchema
```

C# syntax

```
public ULIndexSchema GetIndex(string name)
```


Parameters

- **name** The name of the index.

Returns

A ULIndexSchema object representing the named index.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ULIndexSchema class" on page 283

GetIndexName method

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

Visual Basic syntax

```
Public Function GetIndexName(ByVal indexID As Integer) As String
```

C# syntax

```
public string GetIndexName(int indexID)
```

Parameters

- **indexID** The ID of the index. The value must be in the range [1,IndexCount].

Returns

The name of the index as a string.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

Index IDs and counts may change during a schema upgrade. To correctly identify an index, access it by name or refresh the cached IDs and counts after a schema upgrade.

See also

- "IndexCount property" on page 428

GetOptimalIndex method

The optimal index for searching a table using the specified column.

Visual Basic syntax

```
Public Function GetOptimalIndex(ByVal columnID As Integer) As String
```

C# syntax

```
public string GetOptimalIndex(int columnID)
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount>-1]. The first column in the table has an ID value of zero.

Returns

A ULIndexSchema object representing the optimal index for the specified column.

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The specified column is the first column in the index, but the index may have more than one column.

See also

- "ColumnCount property" on page 195
- "ULIndexSchema class" on page 283

GetPublicationPredicate method

Returns the publication predicate for this table in the named publication.

Visual Basic syntax

```
Public Function GetPublicationPredicate(
    ByVal pubName As String
) As String
```

C# syntax

```
public string GetPublicationPredicate(string pubName)
```

Parameters

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

Returns

The publication predicate as a string.

Exceptions

- **"ULException class"** A SQL error occurred.

IsColumnAutoIncrement method

Checks whether the specified column's default is set to autoincrement.

Visual Basic syntax

```
Public Function IsColumnAutoIncrement(
    ByVal columnID As Integer
) As Boolean
```

C# syntax

```
public bool IsColumnAutoIncrement(int columnID)
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

True if the column is autoincrementing, false if it is not autoincrementing.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ColumnCount property" on page 195

IsColumnCurrentDate method

Checks whether the specified column's default is set to the current date (ULDbType.Date).

Visual Basic syntax

```
Public Function IsColumnCurrentDate(
    ByVal columnID As Integer
) As Boolean
```

C# syntax

```
public bool IsColumnCurrentDate(int columnID)
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

True if the column defaults to the current date, false if the column does not default to the current date.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ColumnCount property" on page 195

IsColumnCurrentTime method

Checks whether the specified column's default is set to the current time (ULDbType.Time).

Visual Basic syntax

```
Public Function IsColumnCurrentTime(
    ByVal columnID As Integer
) As Boolean
```

C# syntax

```
public bool IsColumnCurrentTime(int columnID)
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

True if the column defaults to the current time, false if the column does not default to the current time.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ColumnCount property" on page 195

IsColumnCurrentTimestamp method

Checks whether the specified column's default is set to the current timestamp (ULDbType.TimeStamp).

Visual Basic syntax

```
Public Function IsColumnCurrentTimestamp(
    ByVal columnID As Integer
) As Boolean
```

C# syntax

```
public bool IsColumnCurrentTimestamp(int columnID)
```


Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

True if the column defaults to the current timestamp, false if the column does not default to the current timestamp.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ColumnCount property" on page 195

IsColumnCurrentUTCTimestamp method

Checks whether the specified column's default is set to the current UTC timestamp (ULDbType.TimeStamp).

Visual Basic syntax

```
Public Function IsColumnCurrentUTCTimestamp(
    ByVal columnID As Integer
) As Boolean
```

C# syntax

```
public bool IsColumnCurrentUTCTimestamp(int columnID)
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

True if the column defaults to the current UTC timestamp, false if the column does not default to the current UTC timestamp.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ColumnCount property" on page 195

IsColumnGlobalAutoIncrement method

Checks whether the specified column's default is set to global autoincrement.

Visual Basic syntax

```
Public Function IsColumnGlobalAutoIncrement(
    ByVal columnID As Integer
) As Boolean
```

C# syntax

```
public bool IsColumnGlobalAutoIncrement(int columnID)
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

True if the column is global autoincrementing, false if it is not global autoincrementing.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "GetColumnPartitionSize method" on page 420
- "DatabaseID property" on page 147
- "ColumnCount property" on page 195

IsColumnNewUUID method

Checks whether the specified column's default is set to a new UUID (System.Guid).

Visual Basic syntax

```
Public Function IsColumnNewUUID(ByVal columnID As Integer) As Boolean
```

C# syntax

```
public bool IsColumnNewUUID(int columnID)
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

True if the column defaults to a new UUID, false if the column does not default to a new UUID.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ColumnCount property" on page 195
- System.Guid

IsColumnNullable method

Checks whether the specified column is nullable.

Visual Basic syntax

```
Public Function IsColumnNullable(ByVal columnID As Integer) As Boolean
```

C# syntax

```
public bool IsColumnNullable(int columnID)
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

True if the column is nullable, false if it is not nullable.

Exceptions

- **"ULException class"** A SQL error occurred.

See also

- "ColumnCount property" on page 195

IsInPublication method

Checks whether the table is contained in the named publication.

Visual Basic syntax

```
Public Function IsInPublication(ByVal pubName As String) As Boolean
```

C# syntax

```
public bool IsInPublication(string pubName)
```

Parameters

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

Returns

True if the table is in the publication, false if the table is not in the publication.

Exceptions

- **"ULException class"** A SQL error occurred.

IndexCount property

Returns the number of indexes on the table.

Visual Basic syntax

```
Public ReadOnly Property IndexCount As Integer
```

C# syntax

```
public int IndexCount {get;}
```

Remarks

The number of indexes on the table or 0 if the table schema is closed.

Index IDs range from 1 to IndexCount, inclusively.

> **Note**
> Index IDs and count may change during a schema upgrade. To correctly identify an index, access it by name or refresh the cached IDs and counts after a schema upgrade.

IsNeverSynchronized property

Checks whether the table is marked as never being synchronized.

Visual Basic syntax

```
Public ReadOnly Property IsNeverSynchronized As Boolean
```

C# syntax

```
public bool IsNeverSynchronized {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

True if the table is marked as never being synchronized, false otherwise.

Tables marked as never being synchronized are never synchronized, even if they are included in a publication. These tables are sometimes referred to as "no sync" tables.

Name property

Returns the name of the table.

Visual Basic syntax

```
Public ReadOnly Overrides Property Name As String
```

C# syntax

```
public override string Name {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

The name of the table as a string.

PrimaryKey property

Returns the index schema of the primary key for the table.

Visual Basic syntax

```
Public ReadOnly Property PrimaryKey As ULIndexSchema
```

C# syntax

```
public ULIndexSchema PrimaryKey {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

A ULIndexSchema object representing the primary key for the table.

See also

- "ULIndexSchema class" on page 283

UploadUnchangedRows property

Checks whether the database uploads rows that have not changed.

Visual Basic syntax

```
Public ReadOnly Property UploadUnchangedRows As Boolean
```

C# syntax

```
public bool UploadUnchangedRows {get;}
```

Exceptions

- **"ULException class"** A SQL error occurred.

Remarks

True if the table is marked to always upload all rows during synchronization, false if the table is marked to upload only changed rows.

Tables marked as such upload unchanged rows, as well as changed rows, when the table is synchronized. These tables are sometimes referred to as "all sync" tables.

ULTransaction class

Represents a SQL transaction.

Visual Basic syntax

```
Public NotInheritable Class ULTransaction
    Inherits System.Data.Common.DbTransaction
```

C# syntax

```
public sealed class ULTransaction : System.Data.Common.DbTransaction
```

Base classes

- System.Data.Common.DbTransaction

Members

All members of ULTransaction class, including all inherited members.

Name	Description
"Commit method"	Commits the database transaction.
Dispose method (Inherited from System.Data.Common.DbTransaction)	Releases the unmanaged resources used by the System.Data.Common.DbTransaction.
"Rollback method"	Rolls back the transaction's outstanding changes to the database.
"Connection property"	Returns the connection associated with the transaction.

Name	Description
"IsolationLevel property"	Returns the isolation level for the transaction.

Remarks

There is no constructor for ULTransaction. To obtain a ULTransaction object, use the ULConnection.BeginTransaction(). To associate a command with a transaction, use the ULCommand.Transaction.

Once a transaction has been committed or rolled back, the connection reverts to automatically committing all operations as they are executed. To group more operations together, a new transaction must be created.

See also

- "BeginTransaction method" on page 117
- "Transaction property" on page 98
- System.Data.Common.DbTransaction

Commit method

Commits the database transaction.

Visual Basic syntax

```
Public Overrides Sub Commit()
```

C# syntax

```
public override void Commit()
```

Remarks

Once a transaction has been committed or rolled back, the connection reverts to automatically committing all operations as they are executed. To group more operations together, a new transaction must be created.

If Commit() fails due to a database error (for example, a referential integrity error), the transaction remains active. Correct the error and call the Commit() method again or call the ULTransaction.Rollback() to complete the transaction.

See also

- "Rollback method" on page 431

Rollback method

Rolls back the transaction's outstanding changes to the database.

Visual Basic syntax

```
Public Overrides Sub Rollback()
```

C# syntax

```
public override void Rollback()
```

Remarks

Once a transaction has been committed or rolled back, the connection reverts to automatically committing all operations as they are executed. To group more operations together, a new transaction must be created.

See also

- "Commit method" on page 431

Connection property

Returns the connection associated with the transaction.

Visual Basic syntax

```
Public ReadOnly Shadows Property Connection As ULConnection
```

C# syntax

```
public new ULConnection Connection {get;}
```

Remarks

The ULConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

This is the strongly-typed version of System.Data.IDbTransaction.Connection and System.Data.Common.DbCommand.Connection.

See also

- "BeginTransaction method" on page 117
- "ULConnection class" on page 110
- System.Data.IDbTransaction.Connection

IsolationLevel property

Returns the isolation level for the transaction.

Visual Basic syntax

```
Public ReadOnly Overrides Property IsolationLevel As IsolationLevel
```


C# syntax

```
public override IsolationLevel IsolationLevel {get;}
```

Remarks

One of the System.Data.IsolationLevel values. UltraLite.NET only supports System.Data.IsolationLevel.ReadUncommitted.

See also

- "BeginTransaction method" on page 117
- System.Data.IsolationLevel

ULInfoMessageEventHandler delegate

Represents the method that handles the ULConnection.InfoMessage event.

Visual Basic syntax

```
Public Delegate Sub ULInfoMessageEventHandler(
    ByVal obj As Object,
    ByVal args As ULInfoMessageEventArgs
)
```

C# syntax

```
public delegate void ULInfoMessageEventHandler(
    object obj,
    ULInfoMessageEventArgs args
);
```

Parameters

- **obj** The connection sending the event.
- **args** The ULInfoMessageEventArgs object that contains the event data.

See also

- "InfoMessage event" on page 151
- "ULInfoMessageEventArgs class" on page 290

ULRowUpdatedEventHandler delegate

Represents the method that handles the ULDataAdapter.RowUpdated event.

Visual Basic syntax

```
Public Delegate Sub ULRowUpdatedEventHandler(
    ByVal sender As Object,
```

```
        ByVal e As ULRowUpdatedEventArgs
    )
```

C# syntax

```
public delegate void ULRowUpdatedEventHandler(
    object sender,
    ULRowUpdatedEventArgs e
);
```

Parameters

- **sender** The connection sending the event.
- **e** The ULRowUpdatedEventArgs object that contains the event data.

See also

- "RowUpdated event" on page 205
- "ULRowUpdatedEventArgs class" on page 362

ULRowUpdatingEventHandler delegate

Represents the method that handles the ULDataAdapter.RowUpdating event.

Visual Basic syntax

```
Public Delegate Sub ULRowUpdatingEventHandler(
    ByVal sender As Object,
    ByVal e As ULRowUpdatingEventArgs
)
```

C# syntax

```
public delegate void ULRowUpdatingEventHandler(
    object sender,
    ULRowUpdatingEventArgs e
);
```

Parameters

- **sender** The connection sending the event.
- **e** The ULRowUpdatingEventArgs object that contains the event data.

See also

- "RowUpdating event" on page 206
- "ULRowUpdatingEventArgs class" on page 365

ULRowsCopiedEventHandler delegate

Represents the method that handles the ULBulkCopy.ULRowsCopied event.

Visual Basic syntax

```
Public Delegate Sub ULRowsCopiedEventHandler(
    ByVal sender As Object,
    ByVal rowsCopiedEventArgs As ULRowsCopiedEventArgs
)
```

C# syntax

```
public delegate void ULRowsCopiedEventHandler(
    object sender,
    ULRowsCopiedEventArgs rowsCopiedEventArgs
);
```

Remarks

The ULRowsCopiedEventHandler delegate is not available in the .NET Compact Framework 2.0.

See also

- "ULRowsCopied event" on page 49
- System.Object

ULAuthStatusCode enumeration

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

Visual Basic syntax

```
Public Enum ULAuthStatusCode
```

C# syntax

```
public enum ULAuthStatusCode
```

Members

Member name	Description	Value
UNKNOWN	Authorization status is unknown, possibly because the connection has not yet performed a synchronization (UNKNOWN = 0).	0
VALID	User ID and password were valid at time of synchronization (VALID = 1).	1
VAL-ID_BUT_EX-PIRES_SOON	User ID and password were valid at time of synchronization, but expires soon (VALID_BUT_EXPIRES_SOON = 2).	2
EXPIRED	User ID or password has expired - authorization failed (EXPIRED = 3).	3

Member name	Description	Value
INVALID	Bad user ID or password - authorization failed (INVALID = 4).	4
IN_USE	User ID is already in use - authorization failed (IN_USE = 5).	5

See also

- "AuthStatus property" on page 391

ULBulkCopyOptions enumeration

A bitwise flag that specifies one or more options to use with an instance of the ULBulkCopy class.

Visual Basic syntax

```
Public Enum ULBulkCopyOptions
```

C# syntax

```
public enum ULBulkCopyOptions
```

Members

Member name	Description	Value
Default	Specifying only this causes the default behavior to be used.	0x0
KeepIdentity	When specified, the source values to be copied into an identity column are preserved. By default, new identity values are generated in the destination table.	0x1
UseInternalTransaction	When specified, each batch of the bulk-copy operation is executed within a transaction. When not specified, transaction aren't used. If you indicate this option and also provide a ULTransaction object to the constructor, a System.ArgumentException occurs.	0x2

Remarks

The ULBulkCopyOptions class is not available in the .NET Compact Framework 2.0.

The ULBulkCopyOptions enumeration is used when you construct a ULBulkCopy instance to specify how WriteToServer methods behave.

See also

- "ULBulkCopy class" on page 39

ULDBValid enumeration

Enumerates the UltraLite.NET database validation methods

Visual Basic syntax

```
Public Enum ULDBValid
```

C# syntax

```
public enum ULDBValid
```

Members

Member name	Description	Value
EXPRESS_VALIDATE	Do a faster, though less thorough, validation.	0
FULL_VALIDATE	Validate tables, indexes, and all database pages.	1

See also

- "ValidateDatabase method" on page 212
- "ValidateDatabase method" on page 144

ULDateOrder enumeration

UL Ext: Enumerates the date orders that a database can support.

Visual Basic syntax

```
Public Enum ULDateOrder
```

C# syntax

```
public enum ULDateOrder
```

Members

Member name	Description
YMD	Year followed by month, followed by day of the month.
MDY	Month followed by day of the month, followed by year.
DMY	Day of the month followed by month, followed by year.

ULDbType enumeration

Enumerates the UltraLite.NET database data types.

Visual Basic syntax

```
Public Enum ULDbType
```

C# syntax

```
public enum ULDbType
```

Members

Member name	Description	Value
BigInt	Signed 64-bit integer.	5
Binary	Binary data, with a specified maximum length. The enumeration values **Binary** and **VarBinary** are aliases of each other.	15
Bit	1-bit flag.	8
Char	Character data, with a specified length. In UltraLite.NET, this type always supports Unicode characters. The types **Char** and **VarChar** are fully compatible.	0
Date	Date information.	10
DateTime	Timestamp information (date, time). The enumeration values **DateTime** and **TimeStamp** are aliases of each other.	9
Decimal	Exact numerical data, with a specified precision and scale. The enumeration values **Decimal** and **Numeric** are aliases of each other.	14
Double	Double precision floating-point number (8 bytes).	12
Float	Single precision floating-point number (4 bytes). The enumeration values **Float** and **Real** are aliases of each other.	13
Integer	Unsigned 32-bit integer.	1

Member name	Description	Value
LongBinary	Binary data, with variable length.	18
LongVarchar	Character data, with variable length. In UltraLite.NET, this type always supports Unicode characters.	17
Numeric	Exact numerical data, with a specified precision and scale. The enumeration values **Decimal** and **Numeric** are aliases of each other.	14
Real	Single precision floating-point number (4 bytes). The enumeration values **Float** and **Real** are aliases of each other.	13
SmallInt	Signed 16-bit integer.	3
STGeometry	ST Geometry information.	ULNET_TYPE_ST_GEOMETRY
Time	Time information.	11
TimeStamp	Timestamp information (date, time). The enumeration values **DateTime** and **TimeStamp** are aliases of each other.	9
TimeStampWithTimeZone	Timestamp information (date, time) along with the time zone offset.	ULNET_TYPE_TIMESTAMP_WITH_TIME_ZONE
TinyInt	Unsigned 8-bit integer.	7
UniqueIdentifier	Universally Unique Identifier (UUID/GUID).	19
UnsignedBigInt	Unsigned 64-bit integer.	6
UnsignedInt	Unsigned 32-bit integer.	2

Member name	Description	Value
UnsignedSmallInt	Unsigned 16-bit integer.	4
VarBinary	Binary data, with a specified maximum length. The enumeration values **Binary** and **VarBinary** are aliases of each other.	15
VarChar	Character data, with a specified maximum length. In UltraLite.NET, this type always supports Unicode characters. The types **Char** and **VarChar** are fully compatible.	16

Remarks

The table below lists which .NET types are compatible with each ULDbType. In the case of integral types, table columns can always be set using smaller integer types, but can also be set using larger types as long as the actual value is within the range of the type.

ULDbType	Compatible .NET type	C# built-in type	Visual Basic built-in type
Binary, VarBinary	System.Byte[], or System.Guid if size is 16	byte[]	Byte()
Bit	System.Boolean	bool	Boolean
Char, VarChar	System.String	String	String
Date	System.DateTime	DateTime (no built-in type)	Date
Double	System.Double	double	Double
LongBinary	System.Byte[]	byte[]	Byte()
LongVarchar	System.String	String	String
Decimal, Numeric	System.String	decimal	Decimal
Float, Real	System.Single	float	Single
BigInt	System.Int64	long	Long

ULDbType	Compatible .NET type	C# built-in type	Visual Basic built-in type
Integer	System.Int32	int	Integer
SmallInt	System.Int16	short	Short
STGeometry	System.String	String	String
Time	System.TimeSpan	TimeSpan (no built-in type)	TimeSpan (no built-in type)
DateTime, TimeStamp	System.DateTime	DateTime (no built-in type)	Date
TimeStampWithTime-Zone	System.String	String	String
TinyInt	System.Byte	byte	Byte
UnsignedBigInt	System.UInt64	ulong	UInt64 (no built-in type)
UnsignedInt	System.UInt32	uint	UInt32 (no built-in type)
UnsignedSmallInt	System.UInt16	ushort	UInt16 (no built-in type)
UniqueIdentifier	System.Guid	Guid (no built-in type)	Guid (no built-in type)

Binary columns of length 16 are fully compatible with the UniqueIdentifier type.

See also

- "GetFieldType method" on page 235
- "GetDataTypeName method" on page 232
- "GetColumnULDbType method" on page 194
- System.Byte

ULRuntimeType enumeration

UL Ext: Enumerates the types of UltraLite.NET runtimes.

Visual Basic syntax

```
Public Enum ULRuntimeType
```

C# syntax

```
public enum ULRuntimeType
```

Members

Member name	Description	Value
STAND-ALONE_UL	Selects the standalone UltraLite.NET runtime. The standalone runtime accesses databases directly. Databases are accessed more quickly this way, but cannot be shared.	0
UL_EN-GINE_CLI-ENT	Selects the UltraLite engine runtime. The UltraLite.NET engine client communicates with the UltraLite engine to access databases. This means that databases can be shared by different applications.	1

See also

- "RuntimeType property" on page 213

ULSqlProgressState enumeration

UL Ext: Enumerates all the states that can occur while executing SQL pass through scripts.

Visual Basic syntax

```
Public Enum ULSqlProgressState
```

C# syntax

```
public enum ULSqlProgressState
```

Members

Member name	Description	Value
STATE_STARTING	No scripts have been executed yet.	0
STATE_RUNNING_SCRIPT	Currently running a SQL pass through script.	1
STATE_DONE	Scripts have successfully completed.	2
STATE_ERROR	Scripts have completed, but an error occurred.	3

See also

- "ULSqlProgressData class" on page 369

ULStreamType enumeration

UL Ext: Enumerates the types of MobiLink synchronization streams to use for synchronization.

Visual Basic syntax

```
Public Enum ULStreamType
```

C# syntax

```
public enum ULStreamType
```

Members

Member name	Description
TCPIP	Synchronize via TCP/IP.
HTTP	Synchronize via HTTP. The HTTP stream uses TCP/IP as its underlying transport. UltraLite applications act as Web browsers and the MobiLink 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.
HTTPS	Synchronize via HTTPS (HTTP with transport-layer security).
TLS	Synchronize via TCP/IP with transport layer security.

Remarks

For information about configuring specific stream types, see "Network protocol options for UltraLite synchronization streams" [*UltraLite - Database Management and Reference*].

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*].

See also

- "Stream property" on page 380

ULSyncProgressState enumeration

UL Ext: Enumerates all the states that can occur while synchronizing.

Visual Basic syntax

```
Public Enum ULSyncProgressState
```

C# syntax

```
public enum ULSyncProgressState
```

Members

Member name	Description	Value
STATE_STARTING	No synchronization actions have been taken yet.	0
STATE_CONNECTING	The synchronization stream has been built, but is not yet opened.	1
STATE_SENDING_HEADER	The synchronization stream has been opened and the header is about to be sent.	2
STATE_SENDING_TABLE	A table is being sent. Progress can be monitored using ULSyncProgressData.SyncTableIndex and ULSyncProgressData.SyncTableCount.	3
STATE_SENDING_DATA	Data for the current table is being sent. ULSyncProgressData.SentBytes, ULSyncProgressData.SentInserts, ULSyncProgressData.SentUpdates, and ULSyncProgressData.SentDeletes have been updated.	4
STATE_FINISHING_UPLOAD	The upload is completing. The final count of rows sent is included with this event.	5
STATE_RECEIVING_UPLOAD_ACK	An acknowledgement that the upload is complete is being received.	6
STATE_RECEIVING_TABLE	A table is being received. Progress can be monitored using ULSyncProgressData.SyncTableIndex and ULSyncProgressData.SyncTableCount.	7
STATE_RECEIVING_DATA	Data for the current table is being received. ULSyncProgressData.ReceivedBytes, ULSyncProgressData.ReceivedInserts, ULSyncProgressData.ReceivedUpdates, and ULSyncProgressData.ReceivedDeletes have been updated.	8
STATE_COMMITTING_DOWNLOAD	The download is being committed. The final count of rows received is included with this event.	9

Member name	Description	Value
STATE_ROLLING_BACK_DOWNLOAD	Synchronization is rolling back the download because an error was encountered during the download. The error is reported with a subsequent STATE_ERROR progress report.	10
STATE_SENDING_DOWNLOAD_ACK	An acknowledgement that the download is complete is being sent.	11
STATE_DISCONNECTING	The synchronization stream is about to be closed.	12
STATE_DONE	Synchronization has successfully completed.	13
STATE_ERROR	Synchronization has completed, but an error occurred.	14
STATE_CANCELLED	Synchronization has been canceled.	15

See also

- "ULSyncProgressData class" on page 382
- "ReceivedBytes property" on page 384
- "ReceivedInserts property" on page 384
- "ReceivedUpdates property" on page 385
- "ReceivedDeletes property" on page 384
- "SentBytes property" on page 385
- "SentInserts property" on page 386
- "SentUpdates property" on page 386
- "SentDeletes property" on page 386
- "SyncTableIndex property" on page 388
- "SyncTableCount property" on page 387

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

