SYBASE iAnywhere

UltraLite® .NET Programming

February 2009

Version 11.0.1

Copyright and trademarks

Contents

About this book

Subject

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

Audience

This book is intended for .NET application developers who want to take advantage of the performance, resource efficiency, robustness, and security of an UltraLite relational database for data storage and synchronization.

About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in four formats that contain identical information.

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

  If you are using a Microsoft Windows operating system, the online Help is provided in HTML Help (CHM) format. To access the documentation, choose **Start** » **Programs** » **SQL Anywhere 11** » **Documentation** » **Online Books**.

  The administration tools use the same online documentation for their Help features.
- **Eclipse** On Unix platforms, the complete online Help is provided in Eclipse format. To access the documentation, run *sadoc* from the *bin32* or *bin64* directory of your SQL Anywhere 11 installation.
- **DocCommentXchange** DocCommentXchange 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

  Visit http://dcx.sybase.com.
- **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 download Adobe Reader, visit http://get.adobe.com/reader/.

  To access the PDF documentation on Microsoft Windows operating systems, choose **Start** » **Programs** » **SQL Anywhere 11** » **Documentation** » **Online Books - PDF Format**.

  To access the PDF documentation on Unix operating systems, use a web browser to open *install-dir/documentation/en/pdf/index.html*.

About the books in the documentation set

The SQL Anywhere documentation consists of the following books:

- **SQL Anywhere 11 - Introduction** This book introduces SQL Anywhere 11, a comprehensive package that provides data management and data exchange, enabling the rapid development of database-powered applications for server, desktop, mobile, and remote office environments.
- **SQL Anywhere 11 - Changes and Upgrading** This book describes new features in SQL Anywhere 11 and in previous versions of the software.
- **SQL Anywhere Server - Database Administration** This book describes how to run, manage, and configure SQL Anywhere databases. It describes database connections, the database server, database

files, backup procedures, security, high availability, replication with the Replication Server, and administration utilities and options.

- **SQL Anywhere Server - Programming** This book describes how to build and deploy database applications using the C, C++, Java, PHP, Perl, Python, and .NET programming languages such as Visual Basic and Visual C#. A variety of programming interfaces such as ADO.NET and ODBC are described.
- **SQL Anywhere Server - SQL Reference** This book provides reference information for system procedures, and the catalog (system tables and views). It also provides an explanation of the SQL Anywhere implementation of the SQL language (search conditions, syntax, data types, and functions).
- **SQL Anywhere Server - SQL Usage** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- **MobiLink - Getting Started** This book introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.
- **MobiLink - Client Administration** This book describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases. This book also describes the Dbmlsync API, which allows you to integrate synchronization seamlessly into your C++ or .NET client applications.
- **MobiLink - Server Administration** This book describes how to set up and administer MobiLink applications.
- **MobiLink - Server-Initiated Synchronization** This book describes MobiLink server-initiated synchronization, a feature that allows the MobiLink server to initiate synchronization or perform actions on remote devices.
- **QAnywhere** This book describes QAnywhere, which is a messaging platform for mobile, wireless, desktop, and laptop clients.
- **SQL Remote** This book describes the SQL Remote data replication system for mobile computing, which enables sharing of data between a SQL Anywhere consolidated database and many SQL Anywhere remote databases using an indirect link such as email or file transfer.
- **UltraLite - Database Management and Reference** This book introduces the UltraLite database system for small devices.
- **UltraLite - C and C++ Programming** This book describes UltraLite C and C++ programming interfaces. With UltraLite, you can develop and deploy database applications to handheld, mobile, or embedded devices.
- **UltraLite - M-Business Anywhere Programming** This book describes UltraLite for M-Business Anywhere. With UltraLite for M-Business Anywhere you can develop and deploy web-based database applications to handheld, mobile, or embedded devices, running Palm OS, Windows Mobile, or Windows.
- **UltraLite - .NET Programming** This book describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.
- **UltraLiteJ** This book describes UltraLiteJ. With UltraLiteJ, you can develop and deploy database applications in environments that support Java. UltraLiteJ supports BlackBerry smartphones and Java SE environments. UltraLiteJ is based on the iAnywhere UltraLite database product.

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

Documentation conventions

This section lists the conventions used in this documentation.

Operating systems

SQL Anywhere runs on a variety of platforms. In most cases, the software behaves 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 (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 Windows Vista and Windows XP, used primarily on server, desktop, and laptop computers, and Windows Mobile used on mobile devices.

  Unless otherwise specified, when the documentation refers to Windows, it refers to all Windows-based platforms, including Windows Mobile.

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

Directory and file names

In most cases, 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. In most cases, you can 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 *dbsrv11.exe*. On Unix, it is *dbsrv11*.

- ***install-dir*** During the installation process, you choose where to install SQL Anywhere. The environment variable SQLANY11 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 *%SQLANY11%\readme.txt*. On Unix, this is equivalent to *$SQLANY11/readme.txt* or *${SQLANY11}/readme.txt*.

  For more information about the default location of *install-dir*, see "SQLANY11 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 SQLANYSAMP11 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*, from the **Start** menu, choose **Programs** » **SQL Anywhere 11** » **Sample Applications And Projects**.

  For more information about the default location of *samples-dir*, see "SQLANYSAMP11 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)"
```

- **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"
  ```

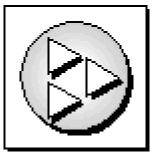
  In many shells, the value of the key would be `my "secret" key`.

- **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}*.

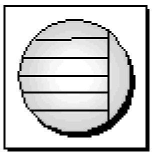
Graphic icons

The following icons are used in this documentation.

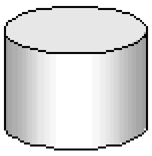
- A client application.



- A database server, such as Sybase SQL Anywhere.



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



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

- A programming interface.

Interface

Contacting the documentation team

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

To submit your comments and suggestions, send an email to the SQL Anywhere documentation team at iasdoc@sybase.com. Although we do not reply to emails, your feedback helps us to improve our documentation, so your input is welcome.

DocCommentXchange

You can also 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

Visit http://dcx.sybase.com.

Finding out more and requesting technical support

Additional information and resources are available at the Sybase iAnywhere Developer Community at http://www.sybase.com/developer/library/sql-anywhere-techcorner.

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: **dbeng11 -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.

Introduction to UltraLite.NET

Contents

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 “RuntimeType property” on page 243 to the appropriate value prior to using any other UltraLite.NET API.

Applications must open a connection to perform operations on a database. Connections are opened using the “ULConnection class” on page 131.

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, then CultureInfo.CurrentCulture, and finally culture **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.
- **ulnet11.dll** This file contains the UltraLite runtime used by a single client. A separate version of the runtime is provided for each target platform.
- **ulnetclient11.dll** This file contains the interface to the UltraLite engine and is used to allow multiple clients to access the engine.

For details 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 2.0 API reference” on page 51.

SQL Anywhere Explorer for UltraLite

Contents

Introduction to the SQL Anywhere Explorer

The SQL Anywhere Explorer is a component that lets you connect to SQL Anywhere and UltraLite databases from Visual Studio.

For information about using the SQL Anywhere Explorer with SQL Anywhere databases, see "SQL Anywhere Explorer" [*SQL Anywhere Server - Programming*].

Using the SQL Anywhere Explorer in UltraLite projects

In Visual Studio, you can use the SQL Anywhere Explorer to create connections to UltraLite databases. Once you connect to a database, you can:

- browse tables and table data
- design programs to open connections with the UltraLite database, or to retrieve and manipulate data
- drag and drop database objects onto C# or Visual Basic code or forms so that the IDE automatically generates code that references the selected object

You can also open Sybase Central and Interactive SQL from Visual Studio by choosing the corresponding command from the **Tools** menu.

Installation note

If you install UltraLite software on a Windows computer that already has Visual Studio installed, the 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 at a command prompt as follows:

- Ensure Visual Studio is not running.
- At a command prompt, run *install-dir\UltraLite\UltraLite.NET\Assembly\v2\installULNet.exe*.

Working with UltraLite database connections in Visual Studio

Use the SQL Anywhere Explorer to display the UltraLite database connections under the **Data Connections** node. You must create a data connection to view the data in the tables.

You can list database tables in the SQL Anywhere Explorer and expand individual tables to list their columns. The properties for an object selected in the SQL Anywhere Explorer window appear in the Visual Studio **Properties** pane.

To add an UltraLite database connection in Visual Studio

1. Open the SQL Anywhere Explorer by choosing **View** » **SQL Anywhere Explorer**.
2. In the SQL Anywhere Explorer window, right-click **Data Connections**, and then choose **Add Connection**.
3. Select **UltraLite**, and then click **OK**.
4. Enter the appropriate values to connect to your database.
5. Click **OK**.

A connection is made to the database, and the connection is added to the **Data Connections** list.

To remove an UltraLite database connection from Visual Studio

1. Open the SQL Anywhere Explorer by choosing **View** » **SQL Anywhere Explorer**.
2. In the SQL Anywhere Explorer window, right-click the UltraLite data connections you want to remove, and then choose **Delete**.

   The connection is removed from the SQL Anywhere Explorer window.

Configuring the SQL Anywhere Explorer

The Visual Studio **Options** window includes settings that you can use to configure the SQL Anywhere Explorer. Some of the options are general options and some are specific to UltraLite usage only.

To access SQL Anywhere Explorer options

1. From the Visual Studio **Tools** menu, choose **Options**.
2. In the left pane of the **Options** window, expand **SQL Anywhere**.
3. Click **General** to configure the SQL Anywhere Explorer general options as required.

   Limit Query Results Sent To Output Window Specify the number of rows that appear in the **Output** window. The default value is 500.

   Show System Objects In Server Explorer Check this option if you want to see system objects in the Microsoft Server Explorer. This is not a SQL Anywhere Explorer option but a Server Explorer option. System objects include those owned by the "dbo" user.

   Sort Objects Choose to sort objects in the for UltraLite Explorer window by object name or by object owner name.

   Generate UI Code When Dropping A Table Or View Onto The Designer Generate the code for tables or views that you drag and drop onto the Windows forms designer.

   Generate Insert, Update, And Delete Commands For Data Adapters Generate INSERT, UPDATE, and DELETE commands for the data adapter when you drag and drop a table or view onto a C# or Visual Basic document.

   Generate Table Mappings For Data Adapters Generate table mappings for the data adapter when you drag and drop a table onto a C# or Visual Basic document.

4. Click UltraLite to configure the specific option for UltraLite.

   Generate When Dropping A Table Into Code Generate code of a specific type for tables that you drag and drop into your application code. Choose from one of the following:

   - ULResultSet represents an editable result set on which you can perform positioned updates and deletes.
   - ULDataReader represents a read-only result set.

- ULTable represents code that allows you to store, remove, update, and read data from a table.
- ULDataAdapter gives you a method to work with table data offline.

Adding UltraLite database objects using the SQL Anywhere Explorer

In Visual Studio, when you drag certain database objects from the SQL Anywhere Explorer and drop them onto Visual Studio designers, the IDE automatically creates new components that reference the selected objects. You can configure the settings for drag and drop operations by choosing **Tools** » **Options** in Visual Studio and opening the SQL Anywhere node.

The following table lists the UltraLite objects you can drag from the SQL Anywhere Explorer, and describes the components created when you drop them onto a Visual Studio Forms Designer or Code Editor.

Item	Result
Data connection	Creates a data connection.
Table	Creates an adapter.

To create a new UltraLite data component using the SQL Anywhere Explorer

1. Open the form or class that you want to add a data component to.
2. In the SQL Anywhere Explorer, select the UltraLite object you want to use.
3. Drag the object from the SQL Anywhere Explorer to the Forms Designer or Code Editor.

Working with UltraLite tables using the SQL Anywhere Explorer

The SQL Anywhere Explorer enables you to view the properties and data for UltraLite tables in database from within Visual Studio.

To view a table in Visual Studio

1. Connect to an UltraLite database using the SQL Anywhere Explorer.
2. In the SQL Anywhere Explorer window, expand your UltraLite database, and then expand Tables.
3. Right-click a table, and then choose **Retrieve Data**.

   The data in the selected table appears in the **Output** window in Visual Studio.

Understanding UltraLite.NET development

Contents

Using SQL Anywhere tools in Visual Studio .NET

Some SQL Anywhere tools are incorporated into Visual Studio 2005 and Visual Studio 2008 to create an integrated database development environment for both SQL Anywhere and UltraLite databases.

- You can use the SQL Anywhere Explorer to view tables and data in those tables, and design programs to open connections to that database for data manipulation and retrieval. The SQL Anywhere Explorer is available from the **View** menu.
- You can open Sybase Central and Interactive SQL without having to leave Visual Studio .NET. Sybase Central and Interactive SQL are available from the **Tools** menu.

Note
For UltraLite development you cannot add adapters for views nor commands for stored procedures; these features are only available to SQL Anywhere databases.

See also

- "Working with UltraLite database connections in Visual Studio" on page 7
- "Adding UltraLite database objects using the SQL Anywhere Explorer" on page 9

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 25.
- **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 28.
- **Synchronization** A set of objects governing synchronization is accessed from the Connection object. See "Synchronization in UltraLite applications" on page 29.
- **Tables** UltraLite tables are accessed using methods of the Connection object. See "Accessing and manipulating data with the Table API" on page 20.
- **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 16.

See "ULConnection class" on page 131.

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 ulcreate 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 131 and "ULConnectionParms class" on page 179.

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 20.

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 86.

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 16.

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 520 and "ULTableSchema class" on page 542.

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 520.

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 25.

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 131 and “ULTransaction class” on page 556.

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:

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

  To obtain a ULDatabaseSchema object, access ULConnection.Schema.

  See "ULConnection class" on page 131.

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

  To obtain a ULTableSchema object, access ULTable.Schema.

- **IndexSchema** 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.

  To obtain a ULIndexSchema object, call the ULTableSchema.GetIndex, the ULTableSchema.GetOptimalIndex, or the ULTableSchema.GetPrimaryKey method.

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

  To obtain a ULPublicationSchema object, call the ULDatabaseSchema.GetPublicationSchema method.

See "ULTableSchema class" on page 542.

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 510
- “ULSyncResult class” on page 514

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 28.

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 131.

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 11 installation.

UltraLite.NET supports TCP/IP, HTTP, HTTPS, and TLS (transport-layer security) synchronization. Synchronization is initiated by the UltraLite application. In all cases, you 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 11 - 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 the Synchronization command 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 53.

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

Contents

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\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**.

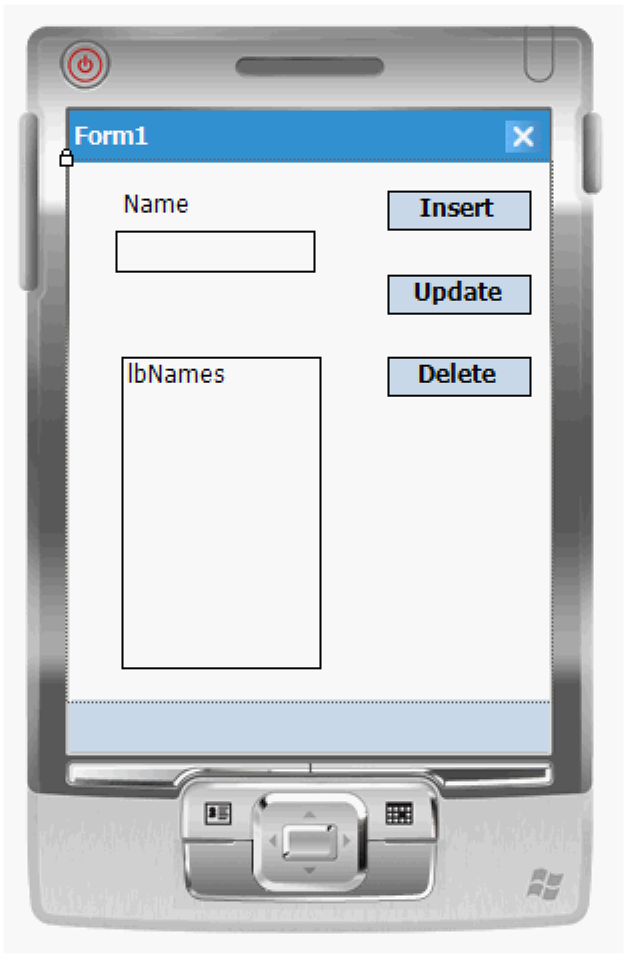
c. Open the folder corresponding to the processor of the Windows Mobile device you are using. For Visual Studio 2005 and later, open the *arm.50* folder. Select *ulnet11.dll*; Click the arrow on the **Add** button and select **Add as Link**.

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 11** » **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 20.

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 *ulnet11.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 38.
- 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 *ulnet11.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 33.
- 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 2.0 API reference

Contents

Namespace

iAnywhere.Data.UltraLite namespace

ULActiveSyncListener interface

UL Ext.: The listener interface for receiving ActiveSync events.

Syntax

Visual Basic
Public Interface **ULActiveSyncListener**

C#
public interface **ULActiveSyncListener**

See also

- "ULActiveSyncListener members" on page 53

ULActiveSyncListener members

Public methods

Member name	Description
"ActiveSyncInvoked method" on page 53	Invoked when the MobiLink provider for ActiveSync calls the application to perform synchronization.

See also

- "ULActiveSyncListener interface" on page 53

ActiveSyncInvoked method

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

Syntax

Visual Basic

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

C#

```
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.

Example

The following code fragments demonstrate how to receive an ActiveSync request and perform a synchronization in the UI thread.

```
' 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 _
      )
    ' Do active sync
    conn.Synchronize()
    ULConnection.DatabaseManager.SignalSyncIsComplete()
  End Sub
End Class


// 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
    // 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();
  }
}
```

See also

- "ULActiveSyncListener interface" on page 53
- "ULActiveSyncListener members" on page 53
- "SignalSyncIsComplete method" on page 249

ULAuthStatusCode enumeration

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

Syntax

Visual Basic
Public Enum **ULAuthStatusCode**

C#
public enum **ULAuthStatusCode**

Members

Member name	Description	Value
EXPIRED	User ID or password has expired - authorization failed (EXPIRED = 3).	3
IN_USE	User ID is already in use - authorization failed (IN_USE = 5).	5
INVALID	Bad user ID or password - authorization failed (INVALID = 4).	4
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
VALID_BUT_EXPIRES_SOON	User ID and password were valid at time of synchronization, but will expire soon (VALID_BUT_EXPIRES_SOON = 2).	2

See also

- "AuthStatus property" on page 515

ULBulkCopy class

Efficiently bulk load an UltraLite table with data from another source. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULBulkCopy**
  Implements IDisposable

C#
public sealed class **ULBulkCopy** : IDisposable

Remarks

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

See also

- "ULBulkCopy members" on page 57

ULBulkCopy members

Public constructors

Member name	Description
"ULBulkCopy constructors" on page 58	Initializes a ULBulkCopy object with the specified ULConnection.

Public properties

Member name	Description
"BatchSize property" on page 61	Gets or sets the number of rows in each batch. At the end of each batch, the rows in the batch are sent to the server.
"BulkCopyTimeout property" on page 62	Gets or sets the number of seconds for the operation to complete before it times out.
"ColumnMappings property" on page 62	Returns a collection of ULBulkCopyColumnMapping items. Column mappings define the relationships between columns in the data source and columns in the destination.
"DestinationTableName property" on page 63	Gets or sets the name of the destination table on the server.
"NotifyAfter property" on page 64	Specifies the number of rows to be processed before generating a notification event.

Public methods

Member name	Description
"Close method" on page 64	Closes the ULBulkCopy instance.
"Dispose method" on page 65	Disposes of the ULBulkCopy instance.
"WriteToServer methods" on page 65	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.

Public events

Member name	Description
"ULRowsCopied event" on page 67	This event occurs every time the number of rows specified by NotifyAfter have been processed.

See also

- "ULBulkCopy class" on page 57

ULBulkCopy constructors

Initializes a ULBulkCopy object with the specified ULConnection.

See also

- "ULConnection class" on page 131

ULBulkCopy(ULConnection) constructor

Syntax

Visual Basic

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

C#

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

Parameters

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

Remarks

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

See also

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57
- "ULBulkCopy constructors" on page 58
- "ULConnection class" on page 131

ULBulkCopy(String) constructor

Initializes a ULBulkCopy object with the specified connection string.

Syntax

Visual Basic

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

C#

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

Parameters

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

  For a list of parameters, see "ConnectionString property" on page 137.

Remarks

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.

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

Implements: System.IDisposable

See also

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57
- "ULBulkCopy constructors" on page 58
- "ULConnectionParms class" on page 179
- IDisposable

ULBulkCopy(String, ULBulkCopyOptions) constructor

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

Syntax

Visual Basic

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

C#

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

Parameters

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

  For a list of parameters, see "ConnectionString property" on page 137.

- **copyOptions** A combination of values from the ULBulkCopyOptions enumeration that determines how data source rows are copied to the destination table.

Remarks

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

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

See also

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57
- "ULBulkCopy constructors" on page 58
- "ULBulkCopyOptions enumeration" on page 85

ULBulkCopy(ULConnection, ULBulkCopyOptions, ULTransaction) constructor

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

Syntax

Visual Basic

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

C#

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

Parameters

- **connection** The already open ULConnection that will be 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 will occur. 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

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

See also

- “ULBulkCopy class” on page 57
- “ULBulkCopy members” on page 57
- “ULBulkCopy constructors” on page 58
- “ULConnection class” on page 131
- “ULTransaction class” on page 556

BatchSize property

Gets or sets the number of rows in each batch. At the end of each batch, the rows in the batch are sent to the server.

Syntax

Visual Basic

```
Public Property BatchSize As Integer
```

C#

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

Property value

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

Remarks

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.

See also

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57

BulkCopyTimeout property

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

Syntax

Visual Basic
Public Property **BulkCopyTimeout** As Integer

C#
public int **BulkCopyTimeout** { get; set; }

Property value

The default value is 30 seconds.

Remarks

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.

See also

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57

ColumnMappings property

Returns a collection of ULBulkCopyColumnMapping items. Column mappings define the relationships between columns in the data source and columns in the destination.

Syntax

Visual Basic
Public Readonly Property **ColumnMappings** As ULBulkCopyColumnMappingCollection

C#
public ULBulkCopyColumnMappingCollection **ColumnMappings** { get;}

Property value

By default, it is an empty collection.

Remarks

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

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57
- "ULBulkCopyColumnMapping class" on page 69

DestinationTableName property

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

Syntax

Visual Basic
Public Property **DestinationTableName** As String

C#
public string **DestinationTableName** { get; set; }

Property value

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

Remarks

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.

See also

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57

NotifyAfter property

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

Syntax

Visual Basic
Public Property **NotifyAfter** As Integer

C#
public int **NotifyAfter** { get; set; }

Property value

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.

Remarks

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

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57
- "BulkCopyTimeout property" on page 62

Close method

Closes the ULBulkCopy instance.

Syntax

Visual Basic
Public Sub **Close()**

C#
public void **Close();**

See also

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57

Dispose method

Disposes of the ULBulkCopy instance.

Syntax

Visual Basic
NotOverridable Public Sub **Dispose()**

C#
public void **Dispose();**

See also

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57

WriteToServer methods

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(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.

Syntax

Visual Basic

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

C#

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

Parameters

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

See also

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57
- "WriteToServer methods" on page 65
- DataRow
- "DestinationTableName property" on page 63

WriteToServer(DataTable) method

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

Syntax

Visual Basic

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

C#

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

Parameters

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

See also

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57
- "WriteToServer methods" on page 65
- "DestinationTableName property" on page 63
- 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.

Syntax

Visual Basic

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

C#

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

Parameters

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

See also

- “ULBulkCopy class” on page 57
- “ULBulkCopy members” on page 57
- “WriteToServer methods” on page 65
- IDataReader
- “DestinationTableName property” on page 63

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.

Syntax

Visual Basic

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

C#

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

Parameters

- **table** A System.Data.DataTable whose rows will 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

- “ULBulkCopy class” on page 57
- “ULBulkCopy members” on page 57
- “WriteToServer methods” on page 65
- “DestinationTableName property” on page 63
- DataTable
- DataRowState

ULRowsCopied event

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

Syntax

Visual Basic
Public Event **ULRowsCopied** As ULRowsCopiedEventHandler

C#
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

- "ULBulkCopy class" on page 57
- "ULBulkCopy members" on page 57
- "NotifyAfter property" on page 64

ULBulkCopyColumnMapping class

Defines the mapping between a column in a ULBulkCopy instance's data source and a column in the instance's destination table. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULBulkCopyColumnMapping**

C#
public sealed class **ULBulkCopyColumnMapping**

Remarks

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

See also

- "ULBulkCopyColumnMapping members" on page 69
- "ULBulkCopy class" on page 57

ULBulkCopyColumnMapping members

Public constructors

Member name	Description
"ULBulkCopyColumnMapping constructors" on page 70	Initializes a new instance of the "ULBulkCopyColumnMapping class" on page 69.

Public properties

Member name	Description
"DestinationColumn property" on page 73	Specifies the name of the column in the destination database table being mapped to.
"DestinationOrdinal property" on page 74	Specifies the ordinal value of the column in the destination database table being mapped to.
"SourceColumn property" on page 74	Specifies the name of the column being mapped in the data source.
"SourceOrdinal property" on page 75	Specifies the ordinal position of the source column within the data source.

See also

- “ULBulkCopyColumnMapping class” on page 69
- “ULBulkCopy class” on page 57

ULBulkCopyColumnMapping constructors

Initializes a new instance of the “ULBulkCopyColumnMapping class” on page 69.

ULBulkCopyColumnMapping() constructor

Creates a new column mapping.

Syntax

Visual Basic

```
Public Sub New()
```

C#

```
public  ULBulkCopyColumnMapping();
```

Remarks

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

See also

- “ULBulkCopyColumnMapping class” on page 69
- “ULBulkCopyColumnMapping members” on page 69
- “ULBulkCopyColumnMapping constructors” on page 70

ULBulkCopyColumnMapping(Int32, Int32) constructor

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

Syntax

Visual Basic

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

C#

```
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

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

See also

- "ULBulkCopyColumnMapping class" on page 69
- "ULBulkCopyColumnMapping members" on page 69
- "ULBulkCopyColumnMapping constructors" on page 70

ULBulkCopyColumnMapping(Int32, 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.

Syntax

Visual Basic

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

C#

```
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

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

See also

- "ULBulkCopyColumnMapping class" on page 69
- "ULBulkCopyColumnMapping members" on page 69
- "ULBulkCopyColumnMapping constructors" on page 70

ULBulkCopyColumnMapping(String, Int32) 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.

Syntax

Visual Basic

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

C#

```
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

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

See also

- "ULBulkCopyColumnMapping class" on page 69
- "ULBulkCopyColumnMapping members" on page 69
- "ULBulkCopyColumnMapping constructors" on page 70

ULBulkCopyColumnMapping(String, String) constructor

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

Syntax

Visual Basic

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

C#

```
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

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

See also

- "ULBulkCopyColumnMapping class" on page 69
- "ULBulkCopyColumnMapping members" on page 69
- "ULBulkCopyColumnMapping constructors" on page 70

DestinationColumn property

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

Syntax

Visual Basic

```
Public Property DestinationColumn As String
```

C#

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

Property value

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.

Remarks

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

- "ULBulkCopyColumnMapping class" on page 69
- "ULBulkCopyColumnMapping members" on page 69
- "DestinationOrdinal property" on page 74
- "DestinationOrdinal property" on page 74

DestinationOrdinal property

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

Syntax

Visual Basic
Public Property **DestinationOrdinal** As Integer

C#
public int **DestinationOrdinal** { get; set; }

Property value

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

Remarks

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

- "ULBulkCopyColumnMapping class" on page 69
- "ULBulkCopyColumnMapping members" on page 69
- "DestinationColumn property" on page 73
- "DestinationColumn property" on page 73

SourceColumn property

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

Syntax

Visual Basic
Public Property **SourceColumn** As String

C#
public string **SourceColumn** { get; set; }

Property value

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.

Remarks

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

- "ULBulkCopyColumnMapping class" on page 69
- "ULBulkCopyColumnMapping members" on page 69
- "SourceOrdinal property" on page 75
- "SourceOrdinal property" on page 75

SourceOrdinal property

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

Syntax

Visual Basic
Public Property **SourceOrdinal** As Integer

C#
public int **SourceOrdinal** { get; set; }

Property value

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

Remarks

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

- "ULBulkCopyColumnMapping class" on page 69
- "ULBulkCopyColumnMapping members" on page 69
- "SourceColumn property" on page 74

ULBulkCopyColumnMappingCollection class

A collection of ULBulkCopyColumnMapping objects that inherits from System.Collections.CollectionBase. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULBulkCopyColumnMappingCollection**
  Inherits CollectionBase

C#
public sealed class **ULBulkCopyColumnMappingCollection**: CollectionBase

Remarks

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

See also

- "ULBulkCopyColumnMappingCollection members" on page 76
- "ULBulkCopyColumnMapping class" on page 69

ULBulkCopyColumnMappingCollection members

Public properties

Member name	Description
Capacity (inherited from CollectionBase)	Gets or sets the number of elements that the CollectionBase can contain.
Count (inherited from CollectionBase)	Gets the number of elements contained in the CollectionBase instance. This property cannot be overwritten.
"Item property" on page 77	Gets the ULBulkCopyColumnMapping object at the specified index.

Public methods

Member name	Description
"Add methods" on page 78	Adds the specified ULBulkCopyColumnMapping to the collection.
Clear (inherited from CollectionBase)	Removes all objects from the CollectionBase instance. This property cannot be overwritten.
"Contains method" on page 81	Returns whether the specified ULBulkCopyColumnMapping object exists in the collection.

Member name	Description
"CopyTo method" on page 82	Copies the elements of the ULBulkCopyColumnMappingCollection to an array of ULBulkCopyColumnMapping items, starting at a particular index.
GetEnumerator (inherited from CollectionBase)	Returns an enumerator that iterates through the CollectionBase instance.
"IndexOf method" on page 83	Returns the index of the specified ULBulkCopyColumnMapping within the collection.
"Remove method" on page 83	Removes the specified ULBulkCopyColumnMapping element from the ULBulkCopyColumnMappingCollection.
"RemoveAt method" on page 84	Removes the mapping at the specified index from the collection.

See also

- "ULBulkCopyColumnMappingCollection class" on page 76
- "ULBulkCopyColumnMapping class" on page 69

Item property

Gets the ULBulkCopyColumnMapping object at the specified index.

Syntax

Visual Basic

```
Public Readonly Property Item( _
  ByVal index As Integer _
) As ULBulkCopyColumnMapping
```

C#

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

Parameters

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

Property value

An ULBulkCopyColumnMapping object is returned.

See also

- “ULBulkCopyColumnMappingCollection class” on page 76
- “ULBulkCopyColumnMappingCollection members” on page 76
- “ULBulkCopyColumnMapping class” on page 69

Add methods

Adds the specified ULBulkCopyColumnMapping to the collection.

Add(ULBulkCopyColumnMapping) method

Adds the specified ULBulkCopyColumnMapping to the collection.

Syntax

Visual Basic

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

C#

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

Parameters

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

Remarks

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

See also

- “ULBulkCopyColumnMappingCollection class” on page 76
- “ULBulkCopyColumnMappingCollection members” on page 76
- “Add methods” on page 78
- “ULBulkCopyColumnMapping class” on page 69

Add(Int32, Int32) method

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

Syntax

Visual Basic
Public Function **Add(** _
  ByVal *sourceColumnOrdinal* As Integer, _
  ByVal *destinationColumnOrdinal* As Integer _
) As ULBulkCopyColumnMapping

C#
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

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

See also

- "ULBulkCopyColumnMappingCollection class" on page 76
- "ULBulkCopyColumnMappingCollection members" on page 76
- "Add methods" on page 78
- "ULBulkCopyColumnMapping class" on page 69

Add(Int32, 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.

Syntax

Visual Basic
Public Function **Add(** _
  ByVal *sourceColumnOrdinal* As Integer, _
  ByVal *destinationColumn* As String _
) As ULBulkCopyColumnMapping

C#
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

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

See also

- "ULBulkCopyColumnMappingCollection class" on page 76
- "ULBulkCopyColumnMappingCollection members" on page 76
- "Add methods" on page 78
- "ULBulkCopyColumnMapping class" on page 69

Add(String, Int32) 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.

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

Syntax

Visual Basic

```
Public Function Add( _
  ByVal sourceColumn As String, _
  ByVal destinationColumnOrdinal As Integer _
) As ULBulkCopyColumnMapping
```

C#

```
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

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

See also

- "ULBulkCopyColumnMappingCollection class" on page 76
- "ULBulkCopyColumnMappingCollection members" on page 76
- "Add methods" on page 78
- "ULBulkCopyColumnMapping class" on page 69

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.

Syntax

Visual Basic

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

C#

```
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

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

See also

- "ULBulkCopyColumnMappingCollection class" on page 76
- "ULBulkCopyColumnMappingCollection members" on page 76
- "Add methods" on page 78
- "ULBulkCopyColumnMapping class" on page 69

Contains method

Returns whether the specified ULBulkCopyColumnMapping object exists in the collection.

Syntax

Visual Basic

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

C#

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

Parameters

- **value** A valid ULBulkCopyColumnMapping object.

Return value

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

See also

- "ULBulkCopyColumnMappingCollection class" on page 76
- "ULBulkCopyColumnMappingCollection members" on page 76
- "ULBulkCopyColumnMapping class" on page 69

CopyTo method

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

Syntax

Visual Basic

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

C#

```
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

- "ULBulkCopyColumnMappingCollection class" on page 76
- "ULBulkCopyColumnMappingCollection members" on page 76
- "ULBulkCopyColumnMapping class" on page 69

IndexOf method

Returns the index of the specified ULBulkCopyColumnMapping within the collection.

Syntax

Visual Basic

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

C#

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

Parameters

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

Return value

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

- "ULBulkCopyColumnMappingCollection class" on page 76
- "ULBulkCopyColumnMappingCollection members" on page 76
- "ULBulkCopyColumnMapping class" on page 69

Remove method

Removes the specified ULBulkCopyColumnMapping element from the ULBulkCopyColumnMappingCollection.

Syntax

Visual Basic

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

C#

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

Parameters

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

See also

- "ULBulkCopyColumnMappingCollection class" on page 76
- "ULBulkCopyColumnMappingCollection members" on page 76
- "ULBulkCopyColumnMapping class" on page 69

RemoveAt method

Removes the mapping at the specified index from the collection.

Syntax

Visual Basic
Public Sub **RemoveAt(** _
  ByVal *index* As Integer _
)

C#
public void **RemoveAt(**
  int *index*
);

Parameters

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

See also

- "ULBulkCopyColumnMappingCollection class" on page 76
- "ULBulkCopyColumnMappingCollection members" on page 76

ULBulkCopyOptions enumeration

A bitwise flag that specifies one or more options to use with an instance of the ULBulkCopy class.

Syntax

Visual Basic
Public Enum **ULBulkCopyOptions**

C#
public enum **ULBulkCopyOptions**

Remarks

The ULBulkCopyOptions enumeration is used when you construct a ULBulkCopy instance to specify how the WriteToServer methods will behave.

Restrictions: The ULBulkCopyOptions class is not available in the .NET Compact Framework 2.0.

Members

Member name	Description	Value
Default	Specifying only this causes the default behavior to be used.	0
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.	1
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.	2

See also

- "ULBulkCopy class" on page 57

ULCommand class

Represents a pre-compiled SQL statement or query, with or without IN parameters. This object can be used to execute a statement or query efficiently multiple times. This class cannot be inherited.

Syntax

Visual Basic

```
Public NotInheritable Class ULCommand
  Inherits DbCommand
  Implements ICloneable
```

C#

```
public sealed class ULCommand: DbCommand,
  ICloneable
```

Remarks

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.
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.

Inherits: System.Data.Common.DbCommand

Implements: System.Data.IDbCommand, System.IDisposable

See also

- "ULCommand members" on page 87
- "CreateCommand method" on page 152
- "Transaction property" on page 98
- "ExecuteNonQuery method" on page 111
- "ExecuteReader() method" on page 112
- "ExecuteResultSet() method" on page 115
- "ExecuteScalar method" on page 117
- "ExecuteTable() method" on page 118
- "ULTable class" on page 520
- "CommandText property" on page 93
- CommandType.TableDirect
- Component.Dispose
- "CommandType property" on page 94
- "IndexName property" on page 96
- DbCommand
- IDbCommand
- IDisposable
- "ULResultSet class" on page 398
- "ULDataReader class" on page 261

ULCommand members

Public constructors

Member name	Description
"ULCommand constructors" on page 90	Initializes a new instance of the "ULCommand class" on page 86.

Public properties

Member name	Description
"CommandText property" on page 93	Specifies the text of the SQL statement or the name of the table when the ULCommand.CommandType is System.Data.CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.

Member name	Description
"CommandTimeout property" on page 94	This feature is not supported by UltraLite.NET.
"CommandType property" on page 94	Specifies the type of command to be executed.
"Connection property" on page 95	The connection object on which to execute the ULCommand object.
"DesignTimeVisible property" on page 96	Indicates if the ULCommand should be visible in a customized Windows Form Designer control.
"IndexName property" on page 96	**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" on page 97	Specifies the parameters for the current statement.
"Plan property" on page 98	**UL Ext.:** Returns the access plan UltraLite.NET uses to execute a query. This property is intended primarily for use during development.
"Transaction property" on page 98	Specifies the ULTransaction in which the ULCommand executes.
"UpdatedRowSource property" on page 99	Specifies how command results are applied to the DataRow when used by the Update method of the ULDataAdapter.

Public methods

Member name	Description
"BeginExecuteNonQuery methods" on page 99	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, given a callback procedure and state information.
"BeginExecuteReader methods" on page 100	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, and retrieves the result set.
"Cancel method" on page 104	This method is not supported in UltraLite.NET.
"CreateParameter method" on page 104	Provides a ULParameter object for supplying parameters to ULCommand objects.

Member name	Description
"EndExecuteNonQuery method" on page 105	Finishes asynchronous execution of a SQL statement.
"EndExecuteReader method" on page 108	Finishes asynchronous execution of a SQL statement, returning the requested ULDataReader.
"ExecuteNonQuery method" on page 111	Executes a statement that does not return a result set, such as a SQL INSERT, DELETE, or UPDATE statement.
"ExecuteReader methods" on page 112	Executes a SQL SELECT statement and returns the result set.
"ExecuteResultSet methods" on page 115	**UL Ext.:** Executes a SQL SELECT statement and returns the result set as a ULResultSet.
"ExecuteScalar method" on page 117	Executes a SQL SELECT statement and returns a single value.
"ExecuteTable methods" on page 118	**UL Ext.:** Retrieves in a ULTable a database table for direct manipulation. The ULCommand.CommandText is interpreted as the name of the table and ULCommand.IndexName can be used to specify a table sorting order.
"Prepare method" on page 120	Pre-compiles and stores the SQL statement of this command.

See also

- "ULCommand class" on page 86
- "CreateCommand method" on page 152
- "Transaction property" on page 98
- "ExecuteNonQuery method" on page 111
- "ExecuteReader() method" on page 112
- "ExecuteResultSet() method" on page 115
- "ExecuteScalar method" on page 117
- "ExecuteTable() method" on page 118
- "ULTable class" on page 520
- "CommandText property" on page 93
- CommandType.TableDirect
- Component.Dispose
- "CommandType property" on page 94
- "IndexName property" on page 96
- DbCommand
- IDbCommand
- IDisposable
- "ULResultSet class" on page 398
- "ULDataReader class" on page 261

ULCommand constructors

Initializes a new instance of the "ULCommand class" on page 86.

ULCommand() constructor

Initializes a ULCommand object.

Syntax

Visual Basic

```
Public Sub New()
```

C#

```
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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "ULCommand constructors" on page 90
- "CreateCommand method" on page 152
- "ULCommand(String) constructor" on page 90
- "ULCommand(String, ULConnection) constructor" on page 91
- "ULCommand(String, ULConnection, ULTransaction) constructor" on page 92
- "CommandText property" on page 93
- "Connection property" on page 95
- "Transaction property" on page 98

ULCommand(String) constructor

Initializes a ULCommand object with the specified command text.

Syntax

Visual Basic

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

C#

```
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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "ULCommand constructors" on page 90
- "CreateCommand method" on page 152
- "ULCommand() constructor" on page 90
- "ULCommand(String, ULConnection) constructor" on page 91
- "ULCommand(String, ULConnection, ULTransaction) constructor" on page 92
- "Connection property" on page 95
- "Transaction property" on page 98
- "CommandType property" on page 94
- CommandType.TableDirect

ULCommand(String, ULConnection) constructor

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

Syntax

Visual Basic

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

C#

```
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

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “ULCommand constructors” on page 90
- “CreateCommand method” on page 152
- “ULCommand() constructor” on page 90
- “ULCommand(String) constructor” on page 90
- “ULCommand(String, ULConnection, ULTransaction) constructor” on page 92
- “Transaction property” on page 98
- “CommandType property” on page 94
- CommandType.TableDirect
- “ULConnection class” on page 131

ULCommand(String, ULConnection, ULTransaction) constructor

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

Syntax

Visual Basic

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

C#

```
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

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “ULCommand constructors” on page 90
- “CreateCommand method” on page 152
- “ULCommand() constructor” on page 90
- “ULCommand(String) constructor” on page 90
- “ULCommand(String, ULConnection) constructor” on page 91
- “CommandType property” on page 94
- CommandType.TableDirect
- “ULTransaction class” on page 556

CommandText property

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

Syntax

Visual Basic
Public Overrides Property **CommandText** As String

C#
public override string **CommandText** { get; set; }

Property value

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

Remarks

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".

Example

The following example demonstrates the use of the parameterized placeholder:

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

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

See also

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “ExecuteNonQuery method” on page 111
- “ExecuteReader() method” on page 112
- “ExecuteResultSet() method” on page 115
- “ExecuteScalar method” on page 117
- “ExecuteTable() method” on page 118
- “CommandType property” on page 94
- CommandType.TableDirect

CommandTimeout property

This feature is not supported by UltraLite.NET.

Syntax

Visual Basic
Public Overrides Property **CommandTimeout** As Integer

C#
public override int **CommandTimeout** { get; set; }

Property value

The value is always zero.

See also

- “ULCommand class” on page 86
- “ULCommand members” on page 87

CommandType property

Specifies the type of command to be executed.

Syntax

Visual Basic
Public Overrides Property **CommandType** As CommandType

C#
public override CommandType **CommandType** { get; set; }

Property value

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

Remarks

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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- CommandType
- CommandType.Text
- CommandType.TableDirect
- "CommandText property" on page 93
- "IndexName property" on page 96
- "ExecuteTable() method" on page 118
- "ExecuteReader() method" on page 112
- CommandType.Text
- "CommandText property" on page 93
- "ExecuteNonQuery method" on page 111
- "ExecuteReader() method" on page 112
- "ExecuteScalar method" on page 117

Connection property

The connection object on which to execute the ULCommand object.

Syntax

Visual Basic
Public Property **Connection** As ULConnection

C#
public ULConnection **Connection** { get; set; }

Property value

The ULConnection object on which to execute the command.

Remarks

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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "ULConnection class" on page 131
- IDbCommand.Connection
- DbCommand.Connection

DesignTimeVisible property

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

Syntax

Visual Basic
Public Overrides Property **DesignTimeVisible** As Boolean

C#
public override bool **DesignTimeVisible** { get; set; }

Property value

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

See also

- "ULCommand class" on page 86
- "ULCommand members" on page 87

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.

Syntax

Visual Basic
Public Property **IndexName** As String

C#
public string **IndexName** { get; set; }

Property value

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

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “ExecuteTable() method” on page 118
- “ExecuteReader() method” on page 112
- “CommandType property” on page 94
- CommandType.TableDirect

Parameters property

Specifies the parameters for the current statement.

Syntax

Visual Basic
Public Readonly Property **Parameters** As ULParameterCollection

C#
public ULParameterCollection **Parameters** { get;}

Property value

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

Remarks

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

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “ULParameterCollection class” on page 375
- “ULParameter class” on page 358
- IDbCommand.Connection
- DbCommand.Connection
- “CommandText property” on page 93

Plan property

UL Ext.: Returns the access plan UltraLite.NET uses to execute a query. This property is intended primarily for use during development.

Syntax

Visual Basic
Public Readonly Property **Plan** As String

C#
public string **Plan** { get;}

Property value

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

See also

- "ULCommand class" on page 86
- "ULCommand members" on page 87

Transaction property

Specifies the ULTransaction in which the ULCommand executes.

Syntax

Visual Basic
Public Property **Transaction** As ULTransaction

C#
public ULTransaction **Transaction** { get; set; }

Property value

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).

Remarks

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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "BeginTransaction() method" on page 145
- "ULTransaction class" on page 556
- "Connection property" on page 95
- IDbCommand.Transaction
- DbCommand.Transaction

UpdatedRowSource property

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

Syntax

Visual Basic
Public Overrides Property **UpdatedRowSource** As UpdateRowSource

C#
public override UpdateRowSource **UpdatedRowSource** { get; set; }

Property value

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

See also

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- UpdateRowSource
- UpdateRowSource.Both

BeginExecuteNonQuery methods

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

BeginExecuteNonQuery() method

Syntax

Visual Basic
Public Function **BeginExecuteNonQuery()** As IAsyncResult

C#
public IAsyncResult **BeginExecuteNonQuery();**

See also

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “BeginExecuteNonQuery methods” on page 99

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.

Syntax

Visual Basic

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

C#

```
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.

Return value

An 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.

See also

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “BeginExecuteNonQuery methods” on page 99
- “EndExecuteNonQuery method” on page 105
- IAsyncResult.AsyncState
- AsyncCallback
- IAsyncResult

BeginExecuteReader methods

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

Syntax

Visual Basic
Public Function **BeginExecuteReader()** As IAsyncResult

C#
public IAsyncResult **BeginExecuteReader();**

Return value

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.

See also

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "BeginExecuteReader methods" on page 100
- "EndExecuteReader method" on page 108
- IAsyncResult
- "ULDataReader class" on page 261

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.

Syntax

Visual Basic
Public Function **BeginExecuteReader(** _
  ByVal *cmdBehavior* As CommandBehavior _
) As IAsyncResult

C#
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.

Return value

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.

See also

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "BeginExecuteReader methods" on page 100
- "EndExecuteReader method" on page 108
- CommandBehavior
- CommandBehavior.Default
- CommandBehavior.CloseConnection
- CommandBehavior.SchemaOnly
- IAsyncResult
- "EndExecuteReader method" on page 108
- "ULDataReader class" on page 261

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.

Syntax

Visual Basic

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

C#

```
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.

Return value

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.

See also

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “BeginExecuteReader methods” on page 100
- “EndExecuteReader method” on page 108
- AsyncCallback
- IAsyncResult.AsyncState
- IAsyncResult.AsyncState
- “EndExecuteReader method” on page 108
- IAsyncResult
- “ULDataReader class” on page 261

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.

Syntax

Visual Basic

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

C#

```
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.

Return value

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.

See also

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "BeginExecuteReader methods" on page 100
- AsyncCallback
- IAsyncResult.AsyncState
- CommandBehavior
- CommandBehavior.Default
- CommandBehavior.CloseConnection
- CommandBehavior.SchemaOnly
- IAsyncResult
- "EndExecuteReader method" on page 108
- "ULDataReader class" on page 261

Cancel method

This method is not supported in UltraLite.NET.

Syntax

Visual Basic
Public Overrides Sub **Cancel()**

C#
public override void **Cancel();**

Remarks

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

See also

- "ULCommand class" on page 86
- "ULCommand members" on page 87

CreateParameter method

Provides a ULParameter object for supplying parameters to ULCommand objects.

Syntax

Visual Basic
Public Function **CreateParameter()** As ULParameter

C#
public ULParameter **CreateParameter();**

Return value

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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "ULParameter class" on page 358
- IDbCommand.CreateParameter
- DbCommand.CreateParameter

EndExecuteNonQuery method

Finishes asynchronous execution of a SQL statement.

Syntax

Visual Basic
Public Function **EndExecuteNonQuery(** _
  ByVal *asyncResult* As IAsyncResult _
) As Integer

C#
public int **EndExecuteNonQuery(**
  IAsyncResult *asyncResult*
);

Parameters

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

Return value

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

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 will block until the command completes
Dim rowCount As Integer = _
  cmd.EndExecuteNonQuery( res )

// C#
ULCommand cmd = new ULCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn
  );
IAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
// this will block 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 )
  ' do other work
End While
' this will block until the command completes
Dim rowCount As Integer = _
  cmd.EndExecuteNonQuery( res )

// C#
ULCommand cmd = new ULCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
```

```
        + " WHERE DepartmentID=100",
        conn
    );
IAsyncResult res = cmd.BeginExecuteNonQuery();
while( !res.IsCompleted ) {
    // do other work
}
// this will block 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 will not block because the command is finished
Dim rowCount As Integer = _
    cmd.EndExecuteNonQuery( res )

// 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 will 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
      ' will be called when the command completes
    End Sub


    // 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
      // will be 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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "BeginExecuteNonQuery() method" on page 99
- IAsyncResult

EndExecuteReader method

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

Syntax

Visual Basic
Public Function **EndExecuteReader(** _
  ByVal *asyncResult* As IAsyncResult _
) As ULDataReader

C#
public ULDataReader **EndExecuteReader(**
  IAsyncResult *asyncResult*
);

Parameters

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

Return value

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

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 will block until the command completes
Dim reader As ULDataReader = _
  cmd.EndExecuteReader( res )

// C#
ULCommand cmd = new ULCommand(
    "SELECT * FROM Departments", conn
  );
IAsyncResult res = cmd.BeginExecuteReader();
// perform other work
// this will block 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 )
```

```
    ' do other work
  End While
  ' this will block 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 ) {
    // do other work
  }
  // this will block 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 will not block because the command is finished
  Dim reader As ULDataReader = _
    cmd.EndExecuteReader( res )

  // C#
  ULCommand cmd = new ULCommand(
      "SELECT * FROM Departments", conn
    );
  IAsyncResult res = cmd.BeginExecuteReader();
  // perform other work
  WaitHandle wh = res.AsyncWaitHandle;
  wh.WaitOne();
  // this will 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
    ' will be called when the command completes
  End Sub


  // 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
    // will be 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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "BeginExecuteReader() method" on page 101
- "ULDataReader class" on page 261
- IAsyncResult

ExecuteNonQuery method

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

Syntax

Visual Basic
Public Overrides Function **ExecuteNonQuery()** As Integer

C#
public override int **ExecuteNonQuery();**

Return value

The number of rows affected.

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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "CommandText property" on page 93
- "Parameters property" on page 97
- "CommandType property" on page 94
- CommandType.TableDirect
- "Connection property" on page 95
- "Transaction property" on page 98
- "CommandText property" on page 93

ExecuteReader methods

Executes a SQL SELECT statement and returns the result set.

ExecuteReader() method

Executes a SQL SELECT statement and returns the result set.

Syntax

Visual Basic
Public Function **ExecuteReader()** As ULDataReader

C#
public ULDataReader **ExecuteReader();**

Return value

The result set as a ULDataReader object.

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

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “ExecuteReader methods” on page 112
- “ExecuteReader(CommandBehavior) method” on page 113
- “CommandText property” on page 93
- “Parameters property” on page 97
- “ULDataReader class” on page 261
- “ExecuteResultSet() method” on page 115
- “ExecuteTable() method” on page 118
- “ULDataAdapter class” on page 229
- “CommandType property” on page 94
- CommandType.TableDirect
- “ExecuteTable() method” on page 118
- “ULTable class” on page 520
- “ULDataReader class” on page 261
- IDbCommand.ExecuteReader
- DbCommand.ExecuteReader
- “Connection property” on page 95
- “Transaction property” on page 98

ExecuteReader(CommandBehavior) method

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

Syntax

Visual Basic
Public Function **ExecuteReader(** _
  ByVal *cmdBehavior* As CommandBehavior _
) As ULDataReader

C#
public 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.

Return value

The result set as a ULDataReader object.

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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "ExecuteReader methods" on page 112
- "ExecuteReader() method" on page 112
- "CommandText property" on page 93
- "Parameters property" on page 97
- "ULDataReader class" on page 261
- "ExecuteResultSet(CommandBehavior) method" on page 116
- "ExecuteTable(CommandBehavior) method" on page 119
- "ULDataAdapter class" on page 229
- "CommandType property" on page 94
- CommandType.TableDirect
- "ExecuteTable(CommandBehavior) method" on page 119
- "ULTable class" on page 520
- "ULDataReader class" on page 261
- IDbCommand.ExecuteReader
- DbCommand.ExecuteReader
- CommandBehavior
- CommandBehavior.Default
- CommandBehavior.CloseConnection
- CommandBehavior.SchemaOnly
- "Connection property" on page 95
- "Transaction property" on page 98
- "CommandText property" on page 93

ExecuteResultSet methods

UL Ext.: Executes a SQL SELECT statement 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.

Syntax

Visual Basic
Public Function **ExecuteResultSet()** As ULResultSet

C#
public ULResultSet **ExecuteResultSet();**

Return value

The result set as a ULResultSet object.

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.

ULCommand.ExecuteResultSet supports positioned updates and deletes with Dynamic SQL.

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" ) && price < 100.0 ) {
        rs.SetDouble( 2, price * .5 );
        rs.Update();
    }
    if( season.Equals( "discontinued" ) ) {
        rs.Delete();
    }
}
rs.Close();
```

See also

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “ExecuteResultSet methods” on page 115
- “ExecuteResultSet(CommandBehavior) method” on page 116
- “ULCommand class” on page 86
- “CommandText property” on page 93
- “Parameters property” on page 97
- “CommandType property” on page 94
- CommandType.TableDirect
- “ULResultSet class” on page 398
- “ULTable class” on page 520

ExecuteResultSet(CommandBehavior) method

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

Syntax

Visual Basic

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

C#

```
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.

Return value

The result set as a ULResultSet object.

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.

ULCommand.ExecuteResultSet supports positioned updates and deletes with Dynamic SQL.

See also

- “ULCommand class” on page 86
- “ULCommand members” on page 87
- “ExecuteResultSet methods” on page 115
- “ExecuteReader() method” on page 112
- “ULResultSet class” on page 398
- “CommandText property” on page 93
- “Parameters property” on page 97
- “ExecuteTable(CommandBehavior) method” on page 119
- “ULDataAdapter class” on page 229
- “CommandType property” on page 94
- CommandType.TableDirect
- “ExecuteTable(CommandBehavior) method” on page 119
- “ULTable class” on page 520
- CommandBehavior
- CommandBehavior.Default
- CommandBehavior.CloseConnection
- CommandBehavior.SchemaOnly
- “Connection property” on page 95
- “Transaction property” on page 98

ExecuteScalar method

Executes a SQL SELECT statement and returns a single value.

Syntax

Visual Basic
Public Overrides Function **ExecuteScalar()** As Object

C#
public override object **ExecuteScalar();**

Return value

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.

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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "Connection property" on page 95
- "Transaction property" on page 98
- "CommandText property" on page 93
- "IndexName property" on page 96
- CommandBehavior
- CommandBehavior.Default
- CommandBehavior.CloseConnection
- CommandBehavior.SchemaOnly
- "CommandType property" on page 94
- "Parameters property" on page 97
- CommandType.TableDirect

ExecuteTable methods

UL Ext.: Retrieves in a ULTable a database table for direct manipulation. The ULCommand.CommandText is interpreted as the name of the table and ULCommand.IndexName can be used to specify a table sorting order.

ExecuteTable() method

UL Ext.: Retrieves in a ULTable a database table for direct manipulation. The ULCommand.CommandText is interpreted as the name of the table and ULCommand.IndexName can be used to specify a table sorting order.

Syntax

Visual Basic
Public Function **ExecuteTable()** As ULTable

C#
public ULTable **ExecuteTable();**

Return value

The table as a ULTable object.

Remarks

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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "ExecuteTable methods" on page 118
- "ExecuteTable(CommandBehavior) method" on page 119
- "Connection property" on page 95
- "Transaction property" on page 98
- "CommandText property" on page 93
- "IndexName property" on page 96
- CommandBehavior
- CommandBehavior.Default
- CommandBehavior.CloseConnection
- CommandBehavior.SchemaOnly
- "CommandType property" on page 94
- CommandType.TableDirect
- "ULTable class" on page 520

ExecuteTable(CommandBehavior) method

UL Ext.: Retrieves, with the specified command behavior, a database table for direct manipulation. The ULCommand.CommandText is interpreted as the name of the table and ULCommand.IndexName can be used to specify a table sorting order.

Syntax

Visual Basic

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

C#

```
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.

Return value

The table as a ULTable object.

Remarks

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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "ExecuteTable methods" on page 118
- "ExecuteTable() method" on page 118
- "Connection property" on page 95
- "Transaction property" on page 98
- "CommandText property" on page 93
- "IndexName property" on page 96
- CommandBehavior
- CommandBehavior.Default
- CommandBehavior.CloseConnection
- CommandBehavior.SchemaOnly
- "CommandType property" on page 94
- CommandType.TableDirect

Prepare method

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

Syntax

Visual Basic
Public Overrides Sub **Prepare()**

C#
public override void **Prepare();**

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

- "ULCommand class" on page 86
- "ULCommand members" on page 87
- "Connection property" on page 95
- "Transaction property" on page 98
- "CommandText property" on page 93

ULCommandBuilder class

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

Syntax

Visual Basic
Public Class **ULCommandBuilder**
  Inherits DbCommandBuilder

C#
public class **ULCommandBuilder**: DbCommandBuilder

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.

Inherits: System.ComponentModel.Component

Implements: System.IDisposable

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)

    'code to modify data in DataSet here

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

    Return dataSet

End Function
```

```
// 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);

           //code to modify data in DataSet here

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

        return dataSet;
    }
}
```

See also

- "ULCommandBuilder members" on page 122
- DataSet
- "ULDataAdapter class" on page 229
- Component
- IDisposable
- "ULCommand class" on page 86
- "ULConnection class" on page 131

ULCommandBuilder members

Public constructors

Member name	Description
"ULCommandBuilder constructors" on page 124	Initializes a new instance of the "ULCommandBuilder class" on page 121.

Public properties

Member name	Description
CatalogLocation (inherited from DbCommandBuilder)	Sets or gets the CatalogLocation for an instance of the DbCommandBuilder.
CatalogSeparator (inherited from DbCommandBuilder)	Sets or gets a string used as the catalog separator for an instance of the DbCommandBuilder.

Member name	Description
ConflictOption (inherited from DbCommandBuilder)	Specifies which ConflictOption is to be used by the DbCommand-Builder.
“DataAdapter property” on page 125	Gets or sets a ULDataAdapter object for which SQL statements are automatically generated.
QuotePrefix (inherited from 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 (inherited from 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 (inherited from DbCommandBuilder)	Gets or sets the character to be used for the separator between the schema identifier and any other identifiers.
SetAllValues (inherited from DbCommandBuilder)	Specifies whether all column values in an update statement are included or only changed ones.

Public methods

Member name	Description
“GetDeleteCommand methods” on page 125	Gets the automatically generated DbCommand object required to perform deletions at the data source.
“GetInsertCommand methods” on page 127	Gets the automatically generated DbCommand object required to perform insertions at the data source.
“GetUpdateCommand methods” on page 128	Gets the automatically generated DbCommand object required to perform updates at the data source.
QuoteIdentifier (inherited from 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 (inherited from DbCommandBuilder)	Clears the commands associated with this DbCommandBuilder.
UnquoteIdentifier (inherited from DbCommandBuilder)	Given a quoted identifier, returns the correct unquoted form of that identifier, including properly un-escaping any embedded quotes in the identifier.

See also

- “ULCommandBuilder class” on page 121
- DataSet
- “ULDataAdapter class” on page 229
- Component
- IDisposable
- “ULCommand class” on page 86
- “ULConnection class” on page 131

ULCommandBuilder constructors

Initializes a new instance of the “ULCommandBuilder class” on page 121.

ULCommandBuilder() constructor

Initializes a ULCommandBuilder object.

Syntax

Visual Basic

Public Sub **New()**

C#

public **ULCommandBuilder();**

See also

- “ULCommandBuilder class” on page 121
- “ULCommandBuilder members” on page 122
- “ULCommandBuilder constructors” on page 124
- “ULCommandBuilder(ULDataAdapter) constructor” on page 124

ULCommandBuilder(ULDataAdapter) constructor

Initializes a ULCommandBuilder object with the specified ULDataAdapter object.

Syntax

Visual Basic

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

C#

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

Parameters

- **adapter** A ULDataAdapter object.

See also

- "ULCommandBuilder class" on page 121
- "ULCommandBuilder members" on page 122
- "ULCommandBuilder constructors" on page 124
- "ULDataAdapter class" on page 229

DataAdapter property

Gets or sets a ULDataAdapter object for which SQL statements are automatically generated.

Syntax

Visual Basic
Public Property **DataAdapter** As ULDataAdapter

C#
public ULDataAdapter **DataAdapter** { get; set; }

Property value

A ULDataAdapter object.

See also

- "ULCommandBuilder class" on page 121
- "ULCommandBuilder members" on page 122
- "ULDataAdapter class" on page 229

GetDeleteCommand methods

Gets the automatically generated DbCommand object required to perform deletions at the data source.

GetDeleteCommand(Boolean) method

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

Syntax

Visual Basic
Public Function **GetDeleteCommand(** _
  ByVal *useColumnsForParameterNames* As Boolean _
) As ULCommand

C#
public ULCommand **GetDeleteCommand(**

  bool *useColumnsForParameterNames*
);

Parameters

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

Return value

The automatically generated ULCommand object required to perform deletions.

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 will still be using 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

- "ULCommandBuilder class" on page 121
- "ULCommandBuilder members" on page 122
- "GetDeleteCommand methods" on page 125
- "ULCommand class" on page 86
- DbCommandBuilder.RefreshSchema
- "SelectCommand property" on page 236
- DbDataAdapter.Update
- DbCommandBuilder.DataAdapter

GetDeleteCommand() method

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

Syntax

Visual Basic
Public Function **GetDeleteCommand()** As ULCommand

C#
public ULCommand **GetDeleteCommand();**

Return value

The automatically generated ULCommand object required to perform deletions.

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 will still be using 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

- "ULCommandBuilder class" on page 121
- "ULCommandBuilder members" on page 122
- "GetDeleteCommand methods" on page 125
- "ULCommand class" on page 86
- DbCommandBuilder.RefreshSchema
- "SelectCommand property" on page 236
- DbDataAdapter.Update
- DbCommandBuilder.DataAdapter

GetInsertCommand methods

Gets the automatically generated DbCommand object required to perform insertions at the data source.

GetInsertCommand(Boolean) method

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

Syntax

Visual Basic

```
Public Function GetInsertCommand( _
  ByVal useColumnsForParameterNames As Boolean _
) As ULCommand
```

C#

```
public ULCommand GetInsertCommand(
  bool useColumnsForParameterNames
);
```

Parameters

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

Return value

The automatically generated ULCommand object required to perform insertions.

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 will still be using 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

- “ULCommandBuilder class” on page 121
- “ULCommandBuilder members” on page 122
- “GetInsertCommand methods” on page 127
- “ULCommand class” on page 86
- DbCommandBuilder.RefreshSchema
- “SelectCommand property” on page 236
- DbDataAdapter.Update
- DbCommandBuilder.DataAdapter

GetInsertCommand() method

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

Syntax

Visual Basic
Public Function **GetInsertCommand()** As ULCommand

C#
public ULCommand **GetInsertCommand();**

Return value

The automatically generated ULCommand object required to perform insertions.

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 will still be using 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

- “ULCommandBuilder class” on page 121
- “ULCommandBuilder members” on page 122
- “GetInsertCommand methods” on page 127
- “ULCommand class” on page 86
- DbCommandBuilder.RefreshSchema
- “SelectCommand property” on page 236
- “ULCommand class” on page 86
- DbCommandBuilder.DataAdapter

GetUpdateCommand methods

Gets the automatically generated DbCommand object required to perform updates at the data source.

GetUpdateCommand(Boolean) method

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

Syntax

Visual Basic
```
Public Function GetUpdateCommand( _
  ByVal useColumnsForParameterNames As Boolean _
) As ULCommand
```

C#
```
public ULCommand GetUpdateCommand(
  bool useColumnsForParameterNames
);
```

Parameters

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

Return value

The automatically generated ULCommand object required to perform updates.

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 will still be using 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

- "ULCommandBuilder class" on page 121
- "ULCommandBuilder members" on page 122
- "GetUpdateCommand methods" on page 128
- "ULCommand class" on page 86
- DbCommandBuilder.RefreshSchema
- "SelectCommand property" on page 236
- DbDataAdapter.Update
- DbCommandBuilder.DataAdapter

GetUpdateCommand() method

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

Syntax

Visual Basic
```
Public Function GetUpdateCommand() As ULCommand
```

C#
public ULCommand **GetUpdateCommand();**

Return value

The automatically generated ULCommand object required to perform updates.

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 will still be using 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

- "ULCommandBuilder class" on page 121
- "ULCommandBuilder members" on page 122
- "GetUpdateCommand methods" on page 128
- "ULCommand class" on page 86
- DbCommandBuilder.RefreshSchema
- "SelectCommand property" on page 236
- DbDataAdapter.Update
- DbCommandBuilder.DataAdapter

ULConnection class

Represents a connection to an UltraLite.NET database. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULConnection**
  Inherits DbConnection

C#
public sealed class **ULConnection**: DbConnection

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.

Implements: System.Data.IDbConnection, System.IDisposable

See also

- "ULConnection members" on page 131
- "RuntimeType property" on page 243
- "Open method" on page 167
- "Schema property" on page 142
- IDbConnection
- IDisposable

ULConnection members

Public constructors

Member name	Description
"ULConnection constructors" on page 135	Initializes a new instance of the "ULConnection class" on page 131.

Public fields

Member name	Description
"INVALID_DATABASE_ID field" on page 137	**UL Ext.:** A database ID constant indicating that the ULConnection.DatabaseID has not been set. This field is constant and read-only.

Public properties

Member name	Description
"ConnectionString property" on page 137	Specifies the parameters to use for opening a connection to an UltraLite.NET database. The connection string can be supplied using a ULConnectionParms object.
"ConnectionTimeout property" on page 138	This feature is not supported by UltraLite.NET.
"DataSource property" on page 139	This feature is not supported by UltraLite.NET.
"Database property" on page 139	Returns the name of the database to which the connection opens.
"DatabaseID property" on page 140	**UL Ext.:** Specifies the Database ID value to be used for global autoincrement columns.
"DatabaseManager property" on page 140	**UL Ext.:** Provides access to the singleton ULDatabaseManager object.
"GlobalAutoIncrementUsage property" on page 141	**UL Ext.:** Returns the percentage of available global autoincrement values that have been used.
"LastIdentity property" on page 141	**UL Ext.:** Returns the most recent identity value used.
"Schema property" on page 142	**UL Ext.:** Provides access to the schema of the current database associated with this connection.
"ServerVersion property" on page 143	This feature is not supported by UltraLite.NET.
"State property" on page 143	Returns the current state of the connection.
"SyncParms property" on page 143	**UL Ext.:** Specifies the synchronization settings for this connection.
"SyncResult property" on page 144	**UL Ext.:** Returns the results of the last synchronization for this connection.

Public methods

Member name	Description
"BeginTransaction methods" on page 144	Starts a database transaction.
"CancelGetNotification method" on page 147	Cancel any pending get-notification calls on all queues matching the given name.
"ChangeDatabase method" on page 148	Changes the current database for an open ULConnection.
"ChangeEncryptionKey method" on page 148	**UL Ext.:** Changes the database's encryption key to the specified new key.
"ChangePassword method" on page 149	Changes the password for the user indicated in the connection string to the supplied new password.
"Close method" on page 150	Closes the database connection.
"CountUploadRows methods" on page 150	**UL Ext.:** Returns the number of rows that need to be uploaded when the next synchronization takes place.
"CreateCommand method" on page 152	Creates and initializes a ULCommand object associated with this connection and its current transaction. You can use the properties of the ULCommand to control its behavior.
"CreateNotificationQueue method" on page 152	Create an event queue.
"DeclareEvent method" on page 153	Declare an named event.
"DestroyNotificationQueue method" on page 154	Destroy a event queue.
EnlistTransaction (inherited from DbConnection)	Enlists in the specified transaction.
"ExecuteNextSQLPassthroughScript method" on page 155	Execute the next pending SQL pass-through script.
"ExecuteSQLPassthroughScripts method" on page 155	Execute all pending SQL pass-through scripts.
"ExecuteTable methods" on page 156	**UL Ext.:** Retrieves in a ULTable a database table for direct manipulation. The table is opened (sorted) using the table's primary key.

Member name	Description
"GetLastDownloadTime method" on page 160	**UL Ext.:** Returns the time of the most recent download of the specified publication.
"GetNewUUID method" on page 161	**UL Ext.:** Generates a new UUID (System.Guid).
"GetNotification method" on page 162	Block for notification or timeout. Returns event name or null.
"GetNotificationParameter method" on page 163	Get value of a parameter, for an event just read by GetNotification().
"GetSQLPassthroughScriptCount method" on page 163	Get the number of pending SQL pass-through scripts.
"GetSchema methods" on page 164	Returns schema information for the data source of this DbConnection.
"GrantConnectTo method" on page 167	**UL Ext.:** Grants access to an UltraLite database for a user ID with a specified password.
"Open method" on page 167	Opens a connection to a database using the previously-specified connection string.
"RegisterForEvent method" on page 168	Register a queue to get events from an object.
"ResetLastDownloadTime method" on page 169	**UL Ext.:** Resets the time of the most recent download.
"RevokeConnectFrom method" on page 170	**UL Ext.:** Revokes access to an UltraLite database from the specified user ID.
"RollbackPartialDownload method" on page 170	**UL Ext.:** Rolls back outstanding changes to the database from a partial download.
"SendNotification method" on page 170	Send a notification to matching queues. Returns the number of matching queues.
"StartSynchronizationDelete method" on page 171	**UL Ext.:** Marks all subsequent deletes made by this connection for synchronization.
"StopSynchronizationDelete method" on page 172	**UL Ext.:** Prevents delete operations from being synchronized.
"Synchronize methods" on page 172	**UL Ext.:** Synchronize the database using the current ULConnection.SyncParms.

Member name	Description
"TriggerEvent method" on page 174	Trigger an event. Returns the number of notifications sent.
"ValidateDatabase method" on page 175	Performs validation on the current database.

Public events

Member name	Description
"InfoMessage event" on page 176	Occurs when UltraLite.NET sends a warning or an informational message on this connection.
"StateChange event" on page 177	Occurs when this connection changes state.

See also

- "ULConnection class" on page 131
- "RuntimeType property" on page 243
- "Open method" on page 167
- "Schema property" on page 142
- IDbConnection
- IDisposable

ULConnection constructors

Initializes a new instance of the "ULConnection class" on page 131.

ULConnection() constructor

Initializes a ULConnection object. The connection must be opened before you can perform any operations against the database.

Syntax

Visual Basic
Public Sub **New()**

C#
public **ULConnection();**

Remarks

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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ULConnection constructors" on page 135
- "Open method" on page 167
- "ConnectionString property" on page 137

ULConnection(String) constructor

Initializes a ULConnection object with the specified connection string. The connection must be opened before you can perform any operations against the database.

Syntax

Visual Basic
Public Sub **New(** _
  ByVal *connectionString* As String _
)

C#
public **ULConnection(**
  string *connectionString*
);

Parameters

- **connectionString** An UltraLite.NET connection string. A connection string is a semicolon-separated list of keyword-value pairs.

  For a list of parameters, see "ConnectionString property" on page 137.

Remarks

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.

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.DatabaseOnCE = "\UltraLite\MyDatabase.udb"
Dim conn As ULConnection = _
  New ULConnection( openParms.ToString() )
conn.Open()

// C#
```

```
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnCE = @"\UltraLite\MyDatabase.udb";
ULConnection conn = new ULConnection( openParms.ToString() );
conn.Open();
```

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ULConnection constructors" on page 135
- "Open method" on page 167
- "ULConnection() constructor" on page 135
- "RuntimeType property" on page 243
- "ULConnectionParms class" on page 179
- "ConnectionString property" on page 137

INVALID_DATABASE_ID field

UL Ext.: A database ID constant indicating that the ULConnection.DatabaseID has not been set. This field is constant and read-only.

Syntax

Visual Basic
Public Shared **INVALID_DATABASE_ID** As Long

C#
public const long **INVALID_DATABASE_ID**;

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "DatabaseID property" on page 140

ConnectionString property

Specifies the parameters to use for opening a connection to an UltraLite.NET database. The connection string can be supplied using a ULConnectionParms object.

Syntax

Visual Basic
Public Overrides Property **ConnectionString** As String

C#
public override string **ConnectionString** { get; set; }

Property value

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).

Remarks

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 semi-colons (;), begin with a quote, or require leading or trailing whitespace.

If you are not quoting parameter values, make sure that they do not contain semi-colons (;), 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).

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.DatabaseOnCE = "\UltraLite\MyDatabase.udb"
Dim conn As ULConnection = New ULConnection
conn.ConnectionString = openParms.ToString()
conn.Open()

// C#
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnCE = @"\UltraLite\MyDatabase.udb";
ULConnection conn = new ULConnection();
conn.ConnectionString = openParms.ToString();
conn.Open();
```

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "Open method" on page 167
- "ULConnectionParms class" on page 179
- "GrantConnectTo method" on page 167

ConnectionTimeout property

This feature is not supported by UltraLite.NET.

Syntax

Visual Basic
Public Overrides Readonly Property **ConnectionTimeout** As Integer

C#
public override int **ConnectionTimeout** { get;}

Property value

The value is always zero.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131

DataSource property

This feature is not supported by UltraLite.NET.

Syntax

Visual Basic
Public Overrides Readonly Property **DataSource** As String

C#
public override string **DataSource** { get;}

Property value

The value is always the empty string.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131

Database property

Returns the name of the database to which the connection opens.

Syntax

Visual Basic
Public Overrides Readonly Property **Database** As String

C#
public override string **Database** { get;}

Property value

A string containing the name of the database.

Remarks

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.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131

DatabaseID property

UL Ext.: Specifies the Database ID value to be used for global autoincrement columns.

Syntax

Visual Basic
Public Property **DatabaseID** As Long

C#
public long **DatabaseID** { get; set; }

Property value

The Database ID value of the current database.

Remarks

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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "GetDatabaseProperty method" on page 255
- "SetDatabaseOption method" on page 259
- UInt32.MaxValue
- "INVALID_DATABASE_ID field" on page 137

DatabaseManager property

UL Ext.: Provides access to the singleton ULDatabaseManager object.

Syntax

Visual Basic
Public Shared Readonly Property **DatabaseManager** As ULDatabaseManager

C#
public const ULDatabaseManager **DatabaseManager** { get;}

Property value

A reference to the singleton ULDatabaseManager object.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ULDatabaseManager class" on page 242

GlobalAutoIncrementUsage property

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

Syntax

Visual Basic
Public Readonly Property **GlobalAutoIncrementUsage** As Short

C#
public short **GlobalAutoIncrementUsage** { get;}

Property value

The percentage of available global autoincrement values that have been used. It is an integer in the range [0-100], inclusive.

Remarks

If the percentage approaches 100, your application should set a new value for the global database ID using ULConnection.DatabaseID.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ULDatabaseManager class" on page 242
- "DatabaseID property" on page 140

LastIdentity property

UL Ext.: Returns the most recent identity value used.

Syntax

Visual Basic
Public Readonly Property **LastIdentity** As UInt64

C#
public ulong **LastIdentity** { get;}

Property value

The most recently-used identity value as an unsigned long.

Remarks

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.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131

Schema property

UL Ext.: Provides access to the schema of the current database associated with this connection.

Syntax

Visual Basic
Public Readonly Property **Schema** As ULDatabaseSchema

C#
public ULDatabaseSchema **Schema** { get;}

Property value

A reference to the ULDatabaseSchema object representing the schema of the database on which this connection opens.

Remarks

This property is only valid while its connection is open.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ULDatabaseSchema class" on page 251

ServerVersion property

This feature is not supported by UltraLite.NET.

Syntax

Visual Basic
Public Overrides Readonly Property **ServerVersion** As String

C#
public override string **ServerVersion** { get;}

Property value

The value is always the empty string.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131

State property

Returns the current state of the connection.

Syntax

Visual Basic
Public Overrides Readonly Property **State** As ConnectionState

C#
public override ConnectionState **State** { get;}

Property value

System.Data.ConnectionState.Open if the connection is open, System.Data.ConnectionState.Closed if the connection is closed.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "StateChange event" on page 177
- ConnectionState.Open
- ConnectionState.Closed

SyncParms property

UL Ext.: Specifies the synchronization settings for this connection.

Syntax

Visual Basic
Public Readonly Property **SyncParms** As ULSyncParms

C#
public ULSyncParms **SyncParms** { get;}

Property value

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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "Synchronize() method" on page 172
- "SyncResult property" on page 144
- "ULSyncParms class" on page 486

SyncResult property

UL Ext.: Returns the results of the last synchronization for this connection.

Syntax

Visual Basic
Public Readonly Property **SyncResult** As ULSyncResult

C#
public ULSyncResult **SyncResult** { get;}

Property value

A reference to the ULSyncResult object representing the results of the last synchronization for this connection.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "Synchronize() method" on page 172
- "SyncParms property" on page 143
- "SyncResult property" on page 144

BeginTransaction methods

Starts a database transaction.

BeginTransaction() method

Returns a transaction object. Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with ULTransaction.Commit or ULTransaction.Rollback.

Syntax

Visual Basic
Public Function **BeginTransaction()** As ULTransaction

C#
public ULTransaction **BeginTransaction();**

Return value

A ULTransaction object representing the new transaction.

Remarks

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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "BeginTransaction methods" on page 144
- "BeginTransaction(IsolationLevel) method" on page 146
- "Commit method" on page 558
- "Rollback method" on page 559
- "Transaction property" on page 98
- "CreateCommand method" on page 152
- IDbConnection.BeginTransaction
- DbConnection.BeginTransaction

BeginTransaction(IsolationLevel) method

Returns a transaction object with the specified isolation level. Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with ULTransaction.Commit or ULTransaction.Rollback.

Syntax

Visual Basic

```
Public Function BeginTransaction( _
  ByVal isolationLevel As IsolationLevel _
) As ULTransaction
```

C#

```
public ULTransaction BeginTransaction(
  IsolationLevel isolationLevel
);
```

Parameters

- **isolationLevel** The required isolation level for the transaction. UltraLite.NET only supports System.Data.IsolationLevel.ReadUncommitted and ReadCommitted.

Return value

A ULTransaction object representing the new transaction.

Remarks

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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "BeginTransaction methods" on page 144
- "ULTransaction class" on page 556
- "BeginTransaction() method" on page 145
- "Commit method" on page 558
- "Rollback method" on page 559
- "Transaction property" on page 98
- "CreateCommand method" on page 152
- IDbConnection.BeginTransaction
- DbConnection.BeginTransaction
- IsolationLevel

CancelGetNotification method

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

Syntax

Visual Basic

```
Public Function CancelGetNotification( _
  ByVal queueName As String _
) As Integer
```

C#

```
public int CancelGetNotification(
  string  queueName
);
```

Parameters

- **queueName** The expression to match queue names upon.

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

- "CreateNotificationQueue method" on page 152
- "DeclareEvent method" on page 153
- "DestroyNotificationQueue method" on page 154
- "DeclareEvent method" on page 153
- "GetNotification method" on page 162
- "RegisterForEvent method" on page 168
- "SendNotification method" on page 170
- "TriggerEvent method" on page 174
- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "GetNotification method" on page 162
- "ULException class" on page 306

ChangeDatabase method

Changes the current database for an open ULConnection.

Syntax

Visual Basic

```
Public Overrides Sub ChangeDatabase( _
  ByVal connectionString As String _
)
```

C#

```
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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ConnectionString property" on page 137

ChangeEncryptionKey method

UL Ext.: Changes the database's encryption key to the specified new key.

Syntax

Visual Basic
Public Sub **ChangeEncryptionKey(** _
  ByVal *newKey* As String _
)

C#
public void **ChangeEncryptionKey(**
  string *newKey*
);

Parameters

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

Remarks

If the encryption key is lost, it is not possible to open the database.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "EncryptionKey property" on page 188

ChangePassword method

Changes the password for the user indicated in the connection string to the supplied new password.

Syntax

Visual Basic
Public Shared Sub **ChangePassword(** _
  ByVal *connectionString* As String, _
  ByVal *newPassword* As String _
)

C#
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.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131

Close method

Closes the database connection.

Syntax

Visual Basic
Public Overrides Sub **Close()**

C#
public override void **Close();**

Remarks

The Close method rolls back any pending transactions and then closes the connection. An application can call Close multiple times.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131

CountUploadRows methods

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

CountUploadRows(String, UInt32) method

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

Syntax

Visual Basic

```
Public Function CountUploadRows( _
  ByVal pubs As String, _
  ByVal threshold As UInt32 _
) As UInt32
```

C#

```
public uint CountUploadRows(
  string  pubs,
  uint 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.

Return value

The number of rows that need to be uploaded from the specified publication(s).

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "CountUploadRows methods" on page 150
- "CountUploadRows(String, Int64) method" on page 151

CountUploadRows(String, Int64) method

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

Syntax

Visual Basic

```
Public Function CountUploadRows( _
  ByVal pubs As String, _
  ByVal threshold As Long _
) As Long
```

C#

```
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. The threshold value must be in the range [0,0x0ffffffff].

Return value

The number of rows that need to be uploaded from the specified publication(s).

Remarks

This method is provided for languages that do not support the System.UInt32 type natively. Use the other form of this method if your application supports it.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "CountUploadRows methods" on page 150
- "CountUploadRows(String, UInt32) method" on page 150

CreateCommand method

Creates and initializes a ULCommand object associated with this connection and its current transaction. You can use the properties of the ULCommand to control its behavior.

Syntax

Visual Basic
Public Function **CreateCommand()** As ULCommand

C#
public ULCommand **CreateCommand();**

Return value

A new ULCommand object.

Remarks

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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ULCommand class" on page 86
- "CommandText property" on page 93
- IDbConnection.CreateCommand
- DbConnection.CreateCommand

CreateNotificationQueue method

Create an event queue.

Syntax

Visual Basic
```
Public Sub CreateNotificationQueue( _
  ByVal queueName As String, _
  ByVal parameters As String _
)
```

C#
```
public void CreateNotificationQueue(
  string  queueName,
  string  parameters
);
```

Parameters

- **queueName** The name of the new queue.
- **parameters** Creation parameters; currently unused, set to NULL.

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

- "CancelGetNotification method" on page 147
- "DeclareEvent method" on page 153
- "DestroyNotificationQueue method" on page 154
- "DeclareEvent method" on page 153
- "GetNotification method" on page 162
- "RegisterForEvent method" on page 168
- "SendNotification method" on page 170
- "TriggerEvent method" on page 174
- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "DestroyNotificationQueue method" on page 154
- "ULException class" on page 306

DeclareEvent method

Declare an named event.

Syntax

Visual Basic

```
Public Sub DeclareEvent( _
  ByVal eventName As String _
)
```

C#

```
public void DeclareEvent(
  string  eventName
);
```

Parameters

- **eventName** The event name.

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

- "CancelGetNotification method" on page 147
- "CreateNotificationQueue method" on page 152
- "DestroyNotificationQueue method" on page 154
- "DeclareEvent method" on page 153
- "GetNotification method" on page 162
- "RegisterForEvent method" on page 168
- "SendNotification method" on page 170
- "TriggerEvent method" on page 174
- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "CreateNotificationQueue method" on page 152
- "ULException class" on page 306

DestroyNotificationQueue method

Destroy a event queue.

Syntax

Visual Basic

```
Public Sub DestroyNotificationQueue( _
  ByVal queueName As String _
)
```

C#

```
public void DestroyNotificationQueue(
  string  queueName
);
```

Parameters

- **queueName** The name of the queue.

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

- “CancelGetNotification method” on page 147
- “CreateNotificationQueue method” on page 152
- “DeclareEvent method” on page 153
- “GetNotification method” on page 162
- “RegisterForEvent method” on page 168
- “SendNotification method” on page 170
- “TriggerEvent method” on page 174
- “ULConnection class” on page 131
- “ULConnection members” on page 131
- “CreateNotificationQueue method” on page 152
- “ULException class” on page 306

ExecuteNextSQLPassthroughScript method

Execute the next pending SQL pass-through script.

Syntax

Visual Basic
Public Sub **ExecuteNextSQLPassthroughScript()**

C#
public void **ExecuteNextSQLPassthroughScript();**

Remarks

Synchronization may result in several SQL pass-through scripts being downloaded.

See also

- “ExecuteSQLPassthroughScripts method” on page 155
- “GetSQLPassthroughScriptCount method” on page 163
- “ULConnection class” on page 131
- “ULConnection members” on page 131
- “GetSQLPassthroughScriptCount method” on page 163
- “ExecuteSQLPassthroughScripts method” on page 155
- “ULException class” on page 306

ExecuteSQLPassthroughScripts method

Execute all pending SQL pass-through scripts.

Syntax

Visual Basic
Public Sub **ExecuteSQLPassthroughScripts()**

C#
public void **ExecuteSQLPassthroughScripts();**

Remarks

Synchronization may result in several SQL pass-through scripts being downloaded. This method is used to execute all pending scripts. Script progress can be tracked with the ULDatabaseManager.SetGlobalListener() method

See also

- "ExecuteNextSQLPassthroughScript method" on page 155
- "GetSQLPassthroughScriptCount method" on page 163
- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "GetSQLPassthroughScriptCount method" on page 163
- "ExecuteNextSQLPassthroughScript method" on page 155
- "ULException class" on page 306
- "SetGlobalListener method" on page 247

ExecuteTable methods

UL Ext.: Retrieves in a ULTable a database table for direct manipulation. The table is opened (sorted) using the table's primary key.

ExecuteTable(String) method

UL Ext.: Retrieves in a ULTable a database table for direct manipulation. The table is opened (sorted) using the table's primary key.

Syntax

Visual Basic
Public Function **ExecuteTable(** _
  ByVal *tableName* As String _
) As ULTable

C#
public ULTable **ExecuteTable(**
  string *tableName*
);

Parameters

- **tableName** The name of the table to open.

Return value

The table as a ULTable object.

Remarks

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()).

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()
```

```
// 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();
// }
```

See also

- “ULConnection class” on page 131
- “ULConnection members” on page 131
- “ExecuteTable methods” on page 156
- “ExecuteTable(String, String) method” on page 157
- “ExecuteTable(String, String, CommandBehavior) method” on page 159
- “ULTable class” on page 520
- “ExecuteTable() method” on page 118
- “ULCommand class” on page 86

ExecuteTable(String, String) method

UL Ext.: Retrieves in a ULTable a database table for direct manipulation. The table is opened (sorted) using the specified index.

Syntax

Visual Basic
Public Function **ExecuteTable(** _
  ByVal *tableName* As String, _
  ByVal *indexName* As String _
) As ULTable

C#
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.

Return value

The table as a ULTable object.

Remarks

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()).

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()
```

```
// 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();
// }
```

See also

- “ULConnection class” on page 131
- “ULConnection members” on page 131
- “ExecuteTable methods” on page 156
- “ExecuteTable(String) method” on page 156
- “ExecuteTable(String, String, CommandBehavior) method” on page 159
- “ULTable class” on page 520
- “ExecuteTable() method” on page 118
- “ULCommand class” on page 86

ExecuteTable(String, String, CommandBehavior) method

UL Ext.: Retrieves, with the specified command behavior, a database table for direct manipulation. The table is opened (sorted) using the specified index.

Syntax

Visual Basic

```
Public Function ExecuteTable( _
  ByVal tableName As String, _
  ByVal indexName As String, _
  ByVal cmdBehavior As CommandBehavior _
) As ULTable
```

C#

```
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.

Return value

The table as a ULTable object.

Remarks

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()).

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
' 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()
```

```
// C#
ULTable t = conn.ExecuteTable(
    "MyTable", "MyIndex", CommandBehavior.Default
  );
// 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(CommandBehavior.Default);
// }
```

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ExecuteTable methods" on page 156
- "ExecuteTable(String) method" on page 156
- "ExecuteTable(String, String) method" on page 157
- "ULTable class" on page 520
- "ExecuteTable(CommandBehavior) method" on page 119
- "ULCommand class" on page 86
- CommandBehavior
- CommandBehavior.Default
- CommandBehavior.CloseConnection
- CommandBehavior.SchemaOnly

GetLastDownloadTime method

UL Ext.: Returns the time of the most recent download of the specified publication.

Syntax

Visual Basic
Public Function **GetLastDownloadTime(** _

```
  ByVal publication As String _
) As Date
```

C#

```
public DateTime GetLastDownloadTime(
  string  publication
);
```

Parameters

- **publication** The publication to check. For more information, see "ULPublicationSchema class" on page 395.

Return value

The timestamp of the last download.

Remarks

The parameter *publication* is a publication name to check. If the special constant ULPublicationSchema.SYNC_ALL_DB is used, returns the time of the last download of the entire database.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ResetLastDownloadTime method" on page 169
- "SYNC_ALL_DB field" on page 396

GetNewUUID method

UL Ext.: Generates a new UUID (System.Guid).

Syntax

Visual Basic

```
Public Function GetNewUUID() As Guid
```

C#

```
public Guid GetNewUUID();
```

Return value

A new UUID as a System.Guid.

Remarks

This method is provided here because it is not included in the .NET Compact Framework.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- Guid

GetNotification method

Block for notification or timeout. Returns event name or null.

Syntax

Visual Basic

```
Public Function GetNotification( _
  ByVal queueName As String, _
  ByVal wait_ms As Integer _
) As String
```

C#

```
public string  GetNotification(
  string  queueName,
  int wait_ms
);
```

Parameters

- **queueName** The name of the queue which will be waited upon.
- **wait_ms** The time to wait, in milliseconds. Use System.Threading.Timeout.Infinite (-1) for an indefinite wait.

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.

Return null if wait period expired or was canceled; otherwise, returns the event name.

See also

- "CancelGetNotification method" on page 147
- "CreateNotificationQueue method" on page 152
- "DeclareEvent method" on page 153
- "DestroyNotificationQueue method" on page 154
- "DeclareEvent method" on page 153
- "RegisterForEvent method" on page 168
- "SendNotification method" on page 170
- "TriggerEvent method" on page 174
- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "SendNotification method" on page 170
- "GetNotificationParameter method" on page 163
- "CancelGetNotification method" on page 147
- "ULException class" on page 306

GetNotificationParameter method

Get value of a parameter, for an event just read by GetNotification().

Syntax

Visual Basic
Public Function **GetNotificationParameter(** _
  ByVal *queueName* As String, _
  ByVal *parameterName* As String _
) As String

C#
public string **GetNotificationParameter(**
  string *queueName*,
  string *parameterName*
);

Parameters

- **queueName** The name of the queue which will be waited upon.
- **parameterName** The name of the parameter whose value should be returned.

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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "GetNotification method" on page 162
- "ULException class" on page 306

GetSQLPassthroughScriptCount method

Get the number of pending SQL pass-through scripts.

Syntax

Visual Basic
Public Function **GetSQLPassthroughScriptCount()** As Long

C#
public long **GetSQLPassthroughScriptCount();**

Remarks

Synchronization may result in several SQL pass-through scripts being downloaded.

See also

- "ExecuteNextSQLPassthroughScript method" on page 155
- "ExecuteSQLPassthroughScripts method" on page 155
- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ExecuteNextSQLPassthroughScript method" on page 155
- "ExecuteSQLPassthroughScripts method" on page 155
- "ULException class" on page 306

GetSchema methods

Returns schema information for the data source of this DbConnection.

GetSchema() method

Returns the list of supported schema collections.

Syntax

Visual Basic

Public Overrides Function **GetSchema()** As DataTable

C#

public override DataTable **GetSchema();**

Remarks

For a description of the available metadata, see "GetSchema(String, String[]) method" on page 165.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "GetSchema methods" on page 164

GetSchema(String) method

Returns information for the specified metadata collection for this ULConnection.

Syntax

Visual Basic

```
Public Overrides Function GetSchema( _
   ByVal collection As String _
) As DataTable
```

C#

```
public override DataTable GetSchema(
  string  collection
);
```

Parameters

- **collection** Name of the metadata collection. If none provided, MetaDataCollections is used.

Remarks

For a description of the available metadata, see "GetSchema(String, String[]) method" on page 165.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "GetSchema methods" on page 164
- "ULConnection class" on page 131

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.

Syntax

Visual Basic

```
Public Overrides Function GetSchema( _
  ByVal collection As String, _
  ByVal restrictions As String() _
) As DataTable
```

C#

```
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.

Return value

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 will return 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.
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 131
- "ULConnection members" on page 131
- "GetSchema methods" on page 164
- "ULConnection class" on page 131
- "ULMetaDataCollectionNames class" on page 349

GrantConnectTo method

UL Ext.: Grants access to an UltraLite database for a user ID with a specified password.

Syntax

Visual Basic

```
Public Sub GrantConnectTo( _
   ByVal uid As String, _
   ByVal pwd As String _
)
```

C#

```
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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "UserID property" on page 189
- "Password property" on page 188
- "ConnectionString property" on page 137

Open method

Opens a connection to a database using the previously-specified connection string.

Syntax

Visual Basic
Public Overrides Sub **Open()**

C#
public override void **Open();**

Remarks

You should explicitly close or dispose of the connection when you are done with it.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ConnectionString property" on page 137
- "State property" on page 143

RegisterForEvent method

Register a queue to get events from an object.

Syntax

Visual Basic

```
Public Sub RegisterForEvent( _
   ByVal eventName As String, _
   ByVal objectName As String, _
   ByVal queueName As String, _
   ByVal registerNotUnReg As Boolean _
)
```

C#

```
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.

Remarks

Register (a queue) to receive notifications of an event. If no queue name is supplied, the default connection queue is implied, and created if required. 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 will receive notifications of the event - other queues with the same name on different connections will not (unless they are also explicit registered). Throws error if queue or event does not exist

See also

- "CancelGetNotification method" on page 147
- "CreateNotificationQueue method" on page 152
- "DeclareEvent method" on page 153
- "DestroyNotificationQueue method" on page 154
- "DeclareEvent method" on page 153
- "GetNotification method" on page 162
- "SendNotification method" on page 170
- "TriggerEvent method" on page 174
- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "DeclareEvent method" on page 153
- "CreateNotificationQueue method" on page 152
- "ULException class" on page 306

ResetLastDownloadTime method

UL Ext.: Resets the time of the most recent download.

Syntax

Visual Basic

```
Public Sub ResetLastDownloadTime( _
  ByVal pubs As String _
)
```

C#

```
public void ResetLastDownloadTime(
  string  pubs
);
```

Parameters

- **pubs** The set of publications to reset. For more information, see "ULPublicationSchema class" on page 395.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "GetLastDownloadTime method" on page 160

RevokeConnectFrom method

UL Ext.: Revokes access to an UltraLite database from the specified user ID.

Syntax

Visual Basic

```
Public Sub RevokeConnectFrom( _
  ByVal uid As String _
)
```

C#

```
public void RevokeConnectFrom(
  string  uid
);
```

Parameters

- **uid** The user ID whose access to the database is being revoked.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "GrantConnectTo method" on page 167

RollbackPartialDownload method

UL Ext.: Rolls back outstanding changes to the database from a partial download.

Syntax

Visual Basic

```
Public Sub RollbackPartialDownload()
```

C#

```
public void RollbackPartialDownload();
```

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "KeepPartialDownload property" on page 490
- "ResumePartialDownload property" on page 493

SendNotification method

Send a notification to matching queues. Returns the number of matching queues.

Syntax

Visual Basic

Public Function **SendNotification(** _
  ByVal *queueName* As String, _
  ByVal *eventName* As String, _
  ByVal *parameters* As String _
) As Integer

C#

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.

Remarks

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

- "CancelGetNotification method" on page 147
- "CreateNotificationQueue method" on page 152
- "DeclareEvent method" on page 153
- "DestroyNotificationQueue method" on page 154
- "DeclareEvent method" on page 153
- "GetNotification method" on page 162
- "RegisterForEvent method" on page 168
- "TriggerEvent method" on page 174
- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "DeclareEvent method" on page 153
- "RegisterForEvent method" on page 168
- "ULException class" on page 306

StartSynchronizationDelete method

UL Ext.: Marks all subsequent deletes made by this connection for synchronization.

Syntax

Visual Basic
Public Sub **StartSynchronizationDelete()**

C#
public void **StartSynchronizationDelete();**

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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "StopSynchronizationDelete method" on page 172
- "Truncate method" on page 541

StopSynchronizationDelete method

UL Ext.: Prevents delete operations from being synchronized.

Syntax

Visual Basic
Public Sub **StopSynchronizationDelete()**

C#
public void **StopSynchronizationDelete();**

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

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "StartSynchronizationDelete method" on page 171

Synchronize methods

UL Ext.: Synchronize the database using the current ULConnection.SyncParms.

Synchronize() method

UL Ext.: Synchronize the database using the current ULConnection.SyncParms.

Syntax

Visual Basic
Public Sub **Synchronize()**

C#
public void **Synchronize();**

Remarks

A detailed result status is reported in this connection's ULConnection.SyncResult.

See also

- “ULConnection class” on page 131
- “ULConnection members” on page 131
- “Synchronize methods” on page 172
- “Synchronize(ULSyncProgressListener) method” on page 173
- “SyncParms property” on page 143
- “SyncResult property” on page 144

Synchronize(ULSyncProgressListener) method

UL Ext.: Synchronize the database using the current ULConnection.SyncParms with progress events posted to the specified listener.

Syntax

Visual Basic
```
Public Sub Synchronize( _
  ByVal listener As ULSyncProgressListener _
)
```

C#
```
public void Synchronize(
  ULSyncProgressListener listener
);
```

Parameters

- **listener** The object that receives synchronization progress events.

Remarks

Errors during synchronization will be posted as a ULSyncProgressState.STATE_ERROR event, and then thrown as a ULException.

A detailed result status will be reported in this connection's ULConnection.SyncResult.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "Synchronize methods" on page 172
- "ULSyncProgressListener interface" on page 510
- "Synchronize() method" on page 172
- "SyncParms property" on page 143
- "ULSyncProgressState enumeration" on page 512
- "ULException class" on page 306
- "SyncResult property" on page 144

TriggerEvent method

Trigger an event. Returns the number of notifications sent.

Syntax

Visual Basic

```
Public Function TriggerEvent( _
  ByVal eventName As String, _
  ByVal parameters As String _
) As Integer
```

C#

```
public int TriggerEvent(
  string  eventName,
  string  parameters
);
```

Parameters

- **eventName** The event name to be triggered.
- **parameters** Parameters to pass.

Remarks

Trigger an event (and send notification to all registered queues).

Returns the number of event notifications sent.

See also

- "CancelGetNotification method" on page 147
- "CreateNotificationQueue method" on page 152
- "DeclareEvent method" on page 153
- "DestroyNotificationQueue method" on page 154
- "DeclareEvent method" on page 153
- "GetNotification method" on page 162
- "RegisterForEvent method" on page 168
- "SendNotification method" on page 170
- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "DeclareEvent method" on page 153
- "RegisterForEvent method" on page 168
- "ULException class" on page 306

ValidateDatabase method

Performs validation on the current database.

Syntax

Visual Basic
Public Sub **ValidateDatabase(** _
  ByVal *how* As ULDBValid, _
  ByVal *tableName* As String _
)

C#
public void **ValidateDatabase(**
  ULDBValid *how*,
  string *tableName*
);

Parameters

- **how** How to validate the database. For more information, see "ULDBValid enumeration" on page 305.
- **tableName** If null (Nothing in Visual Basic), validate the entire database; otherwise, validate just the named table.

Example

The following code validates the current database

```
' Visual Basic
conn.ValidateDatabase( iAnywhere.Data.UltraLite.ULVF_INDEX, Nothing )
```

```
// C#
conn.ValidateDatabase( iAnywhere.Data.UltraLite.ULVF_INDEX, null )
```

See also

- “ULConnection class” on page 131
- “ULConnection members” on page 131
- “ValidateDatabase method” on page 250
- “ULDBValid enumeration” on page 305

InfoMessage event

Occurs when UltraLite.NET sends a warning or an informational message on this connection.

Syntax

Visual Basic
Public Event **InfoMessage** As ULInfoMessageEventHandler

C#
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.

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

// 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

// C#
```

```
conn.InfoMessage +=
  new ULInfoMessageEventHandler(MyInfoMessageHandler);
```

Event data

- **NativeError** The SQL code corresponding to the informational message or warning returned by the database.
- **Message** The informational or warning message string returned by the database.
- **Source** The name of the ADO.NET data provider returning the message.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- "ULInfoMessageEventHandler delegate" on page 348

StateChange event

Occurs when this connection changes state.

Syntax

Visual Basic
Public Overrides Event **StateChange** As StateChangeEventHandler

C#
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.

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

// 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

// C#
conn.StateChange += new StateChangeEventHandler(MyStateHandler);
```

Event data

- **CurrentState** Gets the new state of the connection. The connection object will be in the new state already when the event is fired.
- **OriginalState** Gets the original state of the connection.

See also

- "ULConnection class" on page 131
- "ULConnection members" on page 131
- StateChangeEventHandler

ULConnectionParms class

UL Ext.: Builds a connection string for opening a connection to an UltraLite database. The frequently-used connection parameters are individual properties on the ULConnectionParms object.

Syntax

Visual Basic
Public Class **ULConnectionParms**
  Inherits Component

C#
public class **ULConnectionParms**: Component

Remarks

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.DatabaseOnCE = "\UltraLite\mydb1.udb"
dbName.DatabaseOnDesktop = "somedir\mydb2.udb"

// C#
ULConnectionParms dbName = new ULConnectionParms();
dbName.DatabaseOnCE = "\\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. For more information, see “AdditionalParms property” on page 182.

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

- “ULConnectionParms members” on page 180
- “Open method” on page 167
- “DropDatabase method” on page 245
- “ToString method” on page 190
- “ULConnection class” on page 131
- “ULConnection(String) constructor” on page 136
- “ConnectionString property” on page 137
- “UserID property” on page 189
- “Password property” on page 188
- “EncryptionKey property” on page 188
- “CacheSize property” on page 186
- “ConnectionName property” on page 186
- “RevokeConnectFrom method” on page 170
- “GrantConnectTo method” on page 167
- “ChangeEncryptionKey method” on page 148

ULConnectionParms members

Public constructors

Member name	Description
“ULConnectionParms constructor” on page 182	Initializes a ULConnectionParms instance with its default values.

Public properties

Member name	Description
"AdditionalParms property" on page 182	Specifies additional parameters as a semicolon-separated list of name=value pairs. These parameters are used less frequently.
"CacheSize property" on page 186	Specifies the size of the cache.
"ConnectionName property" on page 186	Specifies a name for the connection. This is only needed if you create more than one connection to the database.
"DatabaseOnCE property" on page 187	Specifies the path and file name of the UltraLite database on Windows Mobile.
"DatabaseOnDesktop property" on page 187	Specifies the path and file name of the UltraLite database on Windows desktop platforms.
"EncryptionKey property" on page 188	Specifies a key for encrypting the database.
"Password property" on page 188	Specifies the password for the authenticated user.
"UserID property" on page 189	Specifies an authenticated user for the database.

Public methods

Member name	Description
"ToString method" on page 190	Returns the string representation of this instance.

Public events

Member name	Description
"UnusedEvent event" on page 190	Unused.

See also

- “ULConnectionParms class” on page 179
- “Open method” on page 167
- “DropDatabase method” on page 245
- “ToString method” on page 190
- “ULConnection class” on page 131
- “ULConnection(String) constructor” on page 136
- “ConnectionString property” on page 137
- “UserID property” on page 189
- “Password property” on page 188
- “EncryptionKey property” on page 188
- “CacheSize property” on page 186
- “ConnectionName property” on page 186
- “RevokeConnectFrom method” on page 170
- “GrantConnectTo method” on page 167
- “ChangeEncryptionKey method” on page 148

ULConnectionParms constructor

Initializes a ULConnectionParms instance with its default values.

Syntax

Visual Basic
Public Sub **New()**

C#
public **ULConnectionParms();**

See also

- “ULConnectionParms class” on page 179
- “ULConnectionParms members” on page 180

AdditionalParms property

Specifies additional parameters as a semicolon-separated list of name=value pairs. These are less commonly used parameters.

Syntax

Visual Basic
Public Property **AdditionalParms** As String

C#
public string **AdditionalParms** { get; set; }

Property value

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).

Remarks

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 (DatabaseOnCE 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.

Keyword	Description
ordered_table_scans	Specifies whether SQL queries without ORDER BY clauses should perform ordered table scans by default. As of release 10.0.1, when using dynamic SQL in UltraLite, if order is not important for executing a query, UltraLite will access 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 ordered_table_scans parameter to 1 (true, yes, on) to revert to the old behavior when iterating over a table. When ordered_table_scans 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 will default to using the primary key. The following parameter string ensures that UltraLite behaves as in previous releases. `createParms.AdditionalParms = "ordered_table_scans=yes"` The default is 0 (false, off, no). This parameter is only used when opening a connection with ULConnection.Open. For more information, see "UltraLite connection parameters" [*UltraLite - Database Management and Reference*].

Keyword	Description
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. Running the database with test data and observing the persistent store file size is recommended. 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 UL-Connection.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 about using the UltraLite engine with Ultra-Lite.NET, see "RuntimeType property" on page 243.

See also

- "ULConnectionParms class" on page 179
- "ULConnectionParms members" on page 180
- "ConnectionString property" on page 137
- "DatabaseOnCE property" on page 187
- "DatabaseOnDesktop property" on page 187
- "Open method" on page 167

CacheSize property

Specifies the size of the cache.

Syntax

Visual Basic
Public Property **CacheSize** As String

C#
public string **CacheSize** { get; set; }

Property value

A string specifying the cache size. The default is a null reference (Nothing in Visual Basic) meaning the default of 16 pages is used.

Remarks

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

For example, the following sets the cache size to 128 KB.

```
connParms.CacheSize = "128k"
```

If the cache size is unspecified or improperly specified, the default size is used. 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.

See also

- “UltraLite CACHE_SIZE connection parameter” [*UltraLite - Database Management and Reference*]
- “ULConnectionParms class” on page 179
- “ULConnectionParms members” on page 180

ConnectionName property

Specifies a name for the connection. This is only needed if you create more than one connection to the database.

Syntax

Visual Basic
Public Property **ConnectionName** As String

C#
public string **ConnectionName** { get; set; }

Property value

A string specifying the name of the connection. The default is a null reference (Nothing in Visual Basic).

See also

- "ULConnectionParms class" on page 179
- "ULConnectionParms members" on page 180

DatabaseOnCE property

Specifies the path and file name of the UltraLite database on Windows Mobile.

Syntax

Visual Basic
Public Property **DatabaseOnCE** As String

C#
public string **DatabaseOnCE** { get; set; }

Property value

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).

See also

- "ULConnectionParms class" on page 179
- "ULConnectionParms members" on page 180

DatabaseOnDesktop property

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

Syntax

Visual Basic
Public Property **DatabaseOnDesktop** As String

C#
public string **DatabaseOnDesktop** { get; set; }

Property value

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).

See also

- “ULConnectionParms class” on page 179
- “ULConnectionParms members” on page 180

EncryptionKey property

Specifies a key for encrypting the database.

Syntax

Visual Basic
Public Property **EncryptionKey** As String

C#
public string **EncryptionKey** { get; set; }

Property value

A string specifying the encryption key. The default is a null reference (Nothing in Visual Basic) meaning no encryption.

Remarks

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

- “ULConnectionParms class” on page 179
- “ULConnectionParms members” on page 180
- “ChangeEncryptionKey method” on page 148

Password property

Specifies the password for the authenticated user.

Syntax

Visual Basic
Public Property **Password** As String

C#
public string **Password** { get; set; }

Property value

A string specifying a database user ID. The default is a null reference (Nothing in Visual Basic).

Remarks

Passwords are case sensitive.

When a database is created, the password for the DBA user ID is set to sql.

See also

- "ULConnectionParms class" on page 179
- "ULConnectionParms members" on page 180
- "UserID property" on page 189

UserID property

Specifies an authenticated user for the database.

Syntax

Visual Basic
Public Property **UserID** As String

C#
public string **UserID** { get; set; }

Property value

A string specifying a database user ID. The default value is a null reference (Nothing in Visual Basic).

Remarks

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

- "ULConnectionParms class" on page 179
- "ULConnectionParms members" on page 180
- "Password property" on page 188
- "GrantConnectTo method" on page 167
- "RevokeConnectFrom method" on page 170

ToString method

Returns the string representation of this instance.

Syntax

Visual Basic
Public Overrides Function **ToString()** As String

C#
public override string **ToString();**

Return value

The string representation of this instance as a semicolon-separated list keyword=value pairs.

See also

- "ULConnectionParms class" on page 179
- "ULConnectionParms members" on page 180

UnusedEvent event

Unused.

Syntax

Visual Basic
Public Event **UnusedEvent** As ULConnectionParms.UnusedEventHandler

C#
public event ULConnectionParms.UnusedEventHandler **UnusedEvent**;

Remarks

This public Event is provided to fix a Visual Studio .NET bug relating to the integration of this class in Visual Basic .NET projects. It has no functional use.

See also

- "ULConnectionParms class" on page 179
- "ULConnectionParms members" on page 180

ULConnectionParms.UnusedEventHandler delegate

UL Ext.: Unused.

Syntax

Visual Basic

```
Public Delegate Sub ULConnectionParms.UnusedEventHandler( _
  ByVal sender As Object, _
  ByVal args As EventArgs _
)
```

C#

```
public delegate void ULConnectionParms.UnusedEventHandler(
  object sender,
  EventArgs args
);
```

Remarks

This public Delegate is provided to fix a Visual Studio .NET bug relating to the integration of this class in Visual Basic .NET projects. It has no functional use.

ULConnectionStringBuilder class

Builds a connection string for opening a connection to an UltraLite database. The frequently-used connection parameters are individual properties on the ULConnectionStringBuilder object. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULConnectionStringBuilder**
  Inherits DbConnectionStringBuilder

C#
public sealed class **ULConnectionStringBuilder**: DbConnectionstring Builder

Remarks

Restrictions: 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.DatabaseOnCE = "\UltraLite\mydb1.udb"
dbName.DatabaseOnDesktop = "somedir\mydb2.udb"

// C#
ULConnectionStringBuilder dbName = new ULConnectionStringBuilder();
dbName.DatabaseOnCE = "\\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. For more information, see “DatabaseName property” on page 199.

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

- “ULConnectionStringBuilder members” on page 193
- “Open method” on page 167
- “DropDatabase method” on page 245
- DbConnectionStringBuilder.ConnectionString
- “ULConnection class” on page 131
- “ULConnection(String) constructor” on page 136
- “ConnectionString property” on page 137
- “DatabaseName property” on page 199
- “UserID property” on page 204
- “Password property” on page 203
- “DatabaseKey property” on page 198
- “CacheSize property” on page 197
- “ConnectionName property” on page 198
- “RevokeConnectFrom method” on page 170
- “GrantConnectTo method” on page 167
- “ChangeEncryptionKey method” on page 148

ULConnectionStringBuilder members

Public constructors

Member name	Description
“ULConnectionStringBuilder constructors” on page 196	Initializes a new instance of the “ULConnectionStringBuilder class” on page 192.

Public properties

Member name	Description
BrowsableConnectionString (inherited from DbConnectionStringBuilder)	Gets or sets a value that indicates whether the DbConnectionStringBuilder.ConnectionString is visible in Visual Studio designers.
"CacheSize property" on page 197	**UL Ext.:** Specifies the size of the cache.
"ConnectionName property" on page 198	Specifies a name for the connection. This is only needed if you create more than one connection to the database.
ConnectionString (inherited from DbConnectionStringBuilder)	Gets or sets the connection string associated with the DbConnectionStringBuilder.
Count (inherited from DbConnectionStringBuilder)	Gets the current number of keys that are contained within the DbConnectionStringBuilder.ConnectionString.
"DatabaseKey property" on page 198	Specifies a key for encrypting the database.
"DatabaseName property" on page 199	Specifies a name for the database or the name of a loaded database to which a connection needs to be made.
"DatabaseOnCE property" on page 199	**UL Ext.:** Specifies the path and file name of the UltraLite database on Windows Mobile.
"DatabaseOnDesktop property" on page 200	**UL Ext.:** Specifies the path and file name of the UltraLite database on Windows desktop platforms.
IsFixedSize (inherited from DbConnectionStringBuilder)	Gets a value that indicates whether the DbConnectionStringBuilder has a fixed size.
IsReadOnly (inherited from DbConnectionStringBuilder)	Gets a value that indicates whether the DbConnectionStringBuilder is read-only.
"Item property" on page 200	Specifies the value of the specified connection keyword.
Keys (inherited from DbConnectionStringBuilder)	Gets an ICollection that contains the keys in the DbConnectionStringBuilder.
"OrderedTableScans property" on page 202	Specifies whether SQL queries without ORDER BY clauses should perform ordered table scans by default.
"Password property" on page 203	Specifies the password for the authenticated user.

Member name	Description
"ReserveSize property" on page 203	**UL Ext.:** Specifies the reserve file system space for storage of UltraLite persistent data.
"StartLine property" on page 204	Specifies the location and then starts the UltraLite engine.
"UserID property" on page 204	Specifies an authenticated user for the database.
Values (inherited from DbConnectionStringBuilder)	Gets an ICollection that contains the values in the DbConnectionStringBuilder.

Public methods

Member name	Description
Add (inherited from DbConnectionStringBuilder)	Adds an entry with the specified key and value into the DbConnectionStringBuilder.
Clear (inherited from DbConnectionStringBuilder)	Clears the contents of the DbConnectionStringBuilder instance.
"ContainsKey method" on page 205	Determines whether the ULConnectionStringBuilder object contains a specific keyword.
"EquivalentTo method" on page 206	Compares the connection information in this ULConnectionStringBuilder object with the connection information in the supplied DbConnectionStringBuilder object.
"GetShortName method" on page 206	Retrieves the short version of the supplied keyword.
"Remove method" on page 207	Removes the entry with the specified key from the ULConnectionStringBuilder instance.
ShouldSerialize (inherited from DbConnectionStringBuilder)	Indicates whether the specified key exists in this DbConnectionStringBuilder instance.
ToString (inherited from DbConnectionStringBuilder)	Returns the connection string associated with this DbConnectionStringBuilder.
"TryGetValue method" on page 207	Retrieves a value corresponding to the supplied key from this ULConnectionStringBuilder.

See also

- "ULConnectionStringBuilder class" on page 192
- "Open method" on page 167
- "DropDatabase method" on page 245
- DbConnectionStringBuilder.ConnectionString
- "ULConnection class" on page 131
- "ULConnection(String) constructor" on page 136
- "ConnectionString property" on page 137
- "DatabaseName property" on page 199
- "UserID property" on page 204
- "Password property" on page 203
- "DatabaseKey property" on page 198
- "CacheSize property" on page 197
- "ConnectionName property" on page 198
- "RevokeConnectFrom method" on page 170
- "GrantConnectTo method" on page 167
- "ChangeEncryptionKey method" on page 148

ULConnectionStringBuilder constructors

Initializes a new instance of the "ULConnectionStringBuilder class" on page 192.

ULConnectionStringBuilder() constructor

Initializes a ULConnectionStringBuilder instance with its default values.

Syntax

Visual Basic
Public Sub **New()**

C#
public **ULConnectionStringBuilder();**

See also

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193
- "ULConnectionStringBuilder constructors" on page 196

ULConnectionStringBuilder(String) constructor

Initializes a ULConnectionStringBuilder instance with the specified connection string.

Syntax

Visual Basic
```
Public Sub New( _
  ByVal connectionString As String _
)
```

C#
```
public  ULConnectionStringBuilder(
  string  connectionString
);
```

Parameters

- **connectionString**   An UltraLite.NET connection string. A connection string is a semicolon-separated list of keyword-value pairs.

See also

- “ULConnectionStringBuilder class” on page 192
- “ULConnectionStringBuilder members” on page 193
- “ULConnectionStringBuilder constructors” on page 196

CacheSize property

UL Ext.: Specifies the size of the cache.

Syntax

Visual Basic
```
Public Property CacheSize As String
```

C#
```
public string  CacheSize { get; set; }
```

Property value

A string specifying the cache size. The default is a null reference (Nothing in Visual Basic) meaning the default of 16 pages is used.

Remarks

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

For example, the following sets the cache size to 128 KB.

```
connParms.CacheSize = "128k"
```

If the cache size is unspecified or improperly specified, the default size is used. 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.

See also

- “UltraLite CACHE_SIZE connection parameter” [*UltraLite - Database Management and Reference*]
- “ULConnectionStringBuilder class” on page 192
- “ULConnectionStringBuilder members” on page 193

ConnectionName property

Specifies a name for the connection. This is only needed if you create more than one connection to the database.

Syntax

Visual Basic
Public Property **ConnectionName** As String

C#
public string **ConnectionName** { get; set; }

Property value

A string specifying the name of the connection. The default is a null reference (Nothing in Visual Basic).

See also

- “ULConnectionStringBuilder class” on page 192
- “ULConnectionStringBuilder members” on page 193

DatabaseKey property

Specifies a key for encrypting the database.

Syntax

Visual Basic
Public Property **DatabaseKey** As String

C#
public string **DatabaseKey** { get; set; }

Property value

A string specifying the encryption key. The default is a null reference (Nothing in Visual Basic) meaning no encryption.

Remarks

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

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193
- "ChangeEncryptionKey method" on page 148

DatabaseName property

Specifies a name for the database or the name of a loaded database to which a connection needs to be made.

Syntax

Visual Basic
Public Property **DatabaseName** As String

C#
public string **DatabaseName** { get; set; }

Property value

A string specifying the name of the database. The default is a null reference (Nothing in Visual Basic).

Remarks

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 (DatabaseOnCE 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

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193
- "DatabaseOnCE property" on page 199
- "DatabaseOnDesktop property" on page 200

DatabaseOnCE property

UL Ext.: Specifies the path and file name of the UltraLite database on Windows Mobile.

Syntax

Visual Basic
Public Property **DatabaseOnCE** As String

C#
public string **DatabaseOnCE** { get; set; }

Property value

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).

See also

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193

DatabaseOnDesktop property

UL Ext.: Specifies the path and file name of the UltraLite database on Windows desktop platforms.

Syntax

Visual Basic
Public Property **DatabaseOnDesktop** As String

C#
public string **DatabaseOnDesktop** { get; set; }

Property value

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).

See also

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193

Item property

Specifies the value of the specified connection keyword.

Syntax

Visual Basic
```
Public Overrides Default Property Item( _
   ByVal keyword As String _
) As Object
```

C#
```
public override object this[
   string  keyword
] { get; set; }
```

Parameters

- **keyword** The name of the connection keyword.

Property value

An object representing the value of the specified connection keyword.

Remarks

Connection keywords and the corresponding properties on ULConnectionStringBuilder are described in the table below:

Keyword	Corresponding Property
cache_size	"CacheSize property" on page 197
ce_file	"DatabaseOnCE property" on page 199
con	"ConnectionName property" on page 198
dbkey	"DatabaseKey property" on page 198
dbn	"DatabaseName property" on page 199
nt_file	"DatabaseOnDesktop property" on page 200
pwd	"Password property" on page 203
reserve_size	"ReserveSize property" on page 203
start	"StartLine property" on page 204
uid	"UserID property" on page 204

See also

- “ULConnectionStringBuilder class” on page 192
- “ULConnectionStringBuilder members” on page 193
- “CacheSize property” on page 197
- “DatabaseOnCE property” on page 199
- “ConnectionName property” on page 198
- “DatabaseKey property” on page 198
- “DatabaseName property” on page 199
- “DatabaseOnDesktop property” on page 200
- “Password property” on page 203
- “ReserveSize property” on page 203
- “StartLine property” on page 204
- “UserID property” on page 204

OrderedTableScans property

Specifies whether SQL queries without ORDER BY clauses should perform ordered table scans by default.

Syntax

Visual Basic
Public Property **OrderedTableScans** As String

C#
public string **OrderedTableScans** { get; set; }

Property value

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).

Remarks

As of release 10.0.1, when using dynamic SQL in UltraLite, if order is not important for executing a query, UltraLite will access 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 will default to using the primary key.

See also

- “ULConnectionStringBuilder class” on page 192
- “ULConnectionStringBuilder members” on page 193

Password property

Specifies the password for the authenticated user.

Syntax

Visual Basic
Public Property **Password** As String

C#
public string **Password** { get; set; }

Property value

A string specifying a database user ID. The default is a null reference (Nothing in Visual Basic).

Remarks

Passwords are case sensitive.

When a database is created, the password for the DBA user ID is set to sql.

See also

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193
- "UserID property" on page 204

ReserveSize property

UL Ext.: Specifies the reserve file system space for storage of UltraLite persistent data.

Syntax

Visual Basic
Public Property **ReserveSize** As String

C#
public string **ReserveSize** { get; set; }

Property value

A string specifying the reserve size. The default is a null reference (Nothing in Visual Basic).

Remarks

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. Running the database with test data and observing the persistent store file size is recommended.

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"
```

See also

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193

StartLine property

Specifies the location and then starts the UltraLite engine.

Syntax

Visual Basic
Public Property **StartLine** As String

C#
public string **StartLine** { get; set; }

Property value

A string specifying the location of the UltraLite engine executable. The default value is a null reference (Nothing in Visual Basic).

Remarks

Only supply a StartLine (START) connection parameter if you are connecting to an engine that is not currently running.

For more information about using the UltraLite engine with UltraLite.NET, see "RuntimeType property" on page 243.

See also

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193
- "RuntimeType property" on page 243

UserID property

Specifies an authenticated user for the database.

Syntax

Visual Basic
Public Property **UserID** As String

C#
public string **UserID** { get; set; }

Property value

A string specifying a database user ID. The default value is a null reference (Nothing in Visual Basic).

Remarks

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

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193
- "Password property" on page 203
- "GrantConnectTo method" on page 167
- "RevokeConnectFrom method" on page 170

ContainsKey method

Determines whether the ULConnectionStringBuilder object contains a specific keyword.

Syntax

Visual Basic
Public Overrides Function **ContainsKey(** _
  ByVal *keyword* As String _
) As Boolean

C#
public override bool **ContainsKey(**
  string *keyword*
);

Parameters

- **keyword** The name of the connection keyword.

Return value

True if this connection string builder contains a value for the specified keyword, otherwise returns false.

See also

- “ULConnectionStringBuilder class” on page 192
- “ULConnectionStringBuilder members” on page 193

EquivalentTo method

Compares the connection information in this ULConnectionStringBuilder object with the connection information in the supplied DbConnectionStringBuilder object.

Syntax

Visual Basic

```
Public Overrides Function EquivalentTo( _
  ByVal connectionStringBuilder As DbConnectionStringBuilder _
) As Boolean
```

C#

```
public override bool EquivalentTo(
  DbConnectionstring Builder connectionStringBuilder
);
```

Parameters

- **connectionStringBuilder** The other DbConnectionStringBuilder object to compare this ULConnectionStringBuilder object to.

Return value

True if this object is equivalent to the specified DbConnectionStringBuilder object, otherwise returns false.

See also

- “ULConnectionStringBuilder class” on page 192
- “ULConnectionStringBuilder members” on page 193
- DbConnectionStringBuilder

GetShortName method

Retrieves the short version of the supplied keyword.

Syntax

Visual Basic

```
Public Shared Function GetShortName( _
  ByVal keyword As String _
) As String
```

C#

```
public static string  GetShortName(
  string  keyword
);
```

Parameters

- **keyword** The key of the item to retrieve.

Return value

The short version of the supplied keyword if keyword is recognized, null otherwise.

See also

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193

Remove method

Removes the entry with the specified key from the ULConnectionStringBuilder instance.

Syntax

Visual Basic

```
Public Overrides Function Remove( _
  ByVal keyword As String _
) As Boolean
```

C#

```
public override bool Remove(
  string  keyword
);
```

Parameters

- **keyword** The name of the connection keyword.

Return value

True if the key existed within the connection string and was removed; false if the key did not exist.

See also

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193

TryGetValue method

Retrieves a value corresponding to the supplied key from this ULConnectionStringBuilder.

Syntax

Visual Basic

```
Public Overrides Function TryGetValue( _
  ByVal keyword As String, _
  ByVal value As Object _
) As Boolean
```

C#

public override bool **TryGetValue(**
   string *keyword*,
   object *value*
);

Parameters

- **keyword** The key of the item to retrieve.
- **value** The value corresponding to the key.

Return value

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 will place the null value (Nothing in Visual Basic) in the value parameter.

See also

- "ULConnectionStringBuilder class" on page 192
- "ULConnectionStringBuilder members" on page 193

ULCreateParms class

UL Ext.: Builds a string of creation-time options for creating an UltraLite database.

Syntax

Visual Basic
Public Class **ULCreateParms**

C#
public class **ULCreateParms**

Remarks

A ULCreateParms object is used to specify the parameters for creating a database (ULDatabaseManager.CreateDatabase(string,byte[],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,byte[],string) method.

For more information, see "UltraLite connection parameters" [*UltraLite - Database Management and Reference*].

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.DatabaseOnCE = "\UltraLite\MyDatabase.udb"
' Assumes file coll_1250LATIN2.vb is
' also compiled in the current project
ULConnection.DatabaseManager.CreateDatabase( _
    openParms.ToString(), _
    iAnywhere.UltraLite.Collations.Collation_1250LATIN2.Data, _
    createParms.ToString() _
  )
Dim conn As ULConnection = _
  New ULConnection( openParms.ToString() )
conn.Open()
```

```
// C#
ULCreateParms createParms = new ULCreateParms();
createParms.CaseSensitive = true;
createParms.UTF8Encoding = true;
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnCE = @"\UltraLite\MyDatabase.udb";
```

```
// Assumes file coll_1250LATIN2.vb is
// also compiled in the current project
ULConnection.DatabaseManager.CreateDatabase(
    openParms.ToString(),
    iAnywhere.UltraLite.Collations.Collation_1250LATIN2.Data,
    createParms.ToString()
  );
ULConnection conn = new ULConnection( openParms.ToString() );
conn.Open();
```

See also

- “ULCreateParms members” on page 210
- “GetDatabaseProperty method” on page 255
- “CreateDatabase method” on page 244
- “ToString method” on page 219

ULCreateParms members

Public constructors

Member name	Description
“ULCreateParms constructor” on page 211	Initializes a ULCreateParms instance with its default values.

Public properties

Member name	Description
“CaseSensitive property” on page 212	Specifies whether the new database should be case sensitive when comparing string values.
“ChecksumLevel property” on page 212	Specifies the level of database page checksums enabled for the new database.
“DateFormat property” on page 213	Specifies the date format used for string conversions by the new database.
“DateOrder property” on page 213	Specifies the date order used for string conversions by the new database.
“FIPS property” on page 214	Specifies whether the new database should be using AES_FIPS encryption or AES encryption.
“MaxHashSize property” on page 214	Specifies the default maximum number of bytes to use for index hashing in the new database.
“NearestCentury property” on page 215	Specifies the nearest century used for string conversions by the new database.

Member name	Description
"Obfuscate property" on page 215	Specifies whether the new database should be using obfuscation (simple encryption) or not.
"PageSize property" on page 216	Specifies the page size of the new database, in bytes or kilobytes.
"Precision property" on page 216	Specifies the floating-point precision used for string conversions by the new database.
"Scale property" on page 216	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" on page 217	Specifies the time format used for string conversions by the new database.
"TimestampFormat property" on page 217	Specifies the timestamp format used for string conversions by the new database.
"TimestampIncrement property" on page 218	Specifies the minimum difference between two unique timestamp values, in microseconds (1,000,000th of a second).
"UTF8Encoding property" on page 218	Specifies whether the new database should be using the UTF8 character set or the character set associated with the collation.

Public methods

Member name	Description
"ToString method" on page 219	Returns the string representation of this instance.

See also

- "ULCreateParms class" on page 209
- "GetDatabaseProperty method" on page 255
- "CreateDatabase method" on page 244
- "ToString method" on page 219

ULCreateParms constructor

Initializes a ULCreateParms instance with its default values.

Syntax

Visual Basic
Public Sub **New()**

C#
public **ULCreateParms();**

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

CaseSensitive property

Specifies whether the new database should be case sensitive when comparing string values.

Syntax

Visual Basic
Public Property **CaseSensitive** As Boolean

C#
public bool **CaseSensitive** { get; set; }

Property value

True if the database should be case sensitive, false if the database should be case insensitive. The default is false.

Remarks

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.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

ChecksumLevel property

Specifies the level of database page checksums enabled for the new database.

Syntax

Visual Basic
Public Property **ChecksumLevel** As Integer

C#
public int **ChecksumLevel** { get; set; }

Property value

An integer specifying the checksum level. Valid values are 0, 1, and 2. The default is 0.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

DateFormat property

Specifies the date format used for string conversions by the new database.

Syntax

Visual Basic
Public Property **DateFormat** As String

C#
public string **DateFormat** { get; set; }

Property value

A string specifying the date format. If the value is a null reference (Nothing in Visual Basic), the database will use "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).

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

DateOrder property

Specifies the date order used for string conversions by the new database.

Syntax

Visual Basic
Public Property **DateOrder** As ULDateOrder

C#
public ULDateOrder **DateOrder** { get; set; }

Property value

A ULDateOrder value identifying the date order for string conversions. The default is YMD.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210
- "ULDateOrder enumeration" on page 300

FIPS property

Specifies whether the new database should be using AES_FIPS encryption or AES encryption.

Syntax

Visual Basic
Public Property **FIPS** As Boolean

C#
public bool **FIPS** { get; set; }

Property value

True if the database should be encrypted using AES_FIPS, false if the database should be encrypted with AES. The default is false.

Remarks

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,byte[],string) method will fail with a missing encryption key error.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210
- "EncryptionKey property" on page 188
- "CreateDatabase method" on page 244

MaxHashSize property

Specifies the default maximum number of bytes to use for index hashing in the new database.

Syntax

Visual Basic
Public Property **MaxHashSize** As Integer

C#
public int **MaxHashSize** { get; set; }

Property value

An integer specifying the maximum hash size. The value must be in the range [0,32]. The default is 8.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

NearestCentury property

Specifies the nearest century used for string conversions by the new database.

Syntax

Visual Basic
Public Property **NearestCentury** As Integer

C#
public int **NearestCentury** { get; set; }

Property value

An integer specifying the nearest century. The value must be in the range [0,100]. The default is 50.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

Obfuscate property

Specifies whether the new database should be using obfuscation (simple encryption) or not.

Syntax

Visual Basic
Public Property **Obfuscate** As Boolean

C#
public bool **Obfuscate** { get; set; }

Property value

True if the database should be encrypted using obfuscation, false if the database should not be obfuscated. The default is false.

Remarks

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 will be ignored.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210
- "FIPS property" on page 214

PageSize property

Specifies the page size of the new database, in bytes or kilobytes.

Syntax

Visual Basic
Public Property **PageSize** As Integer

C#
public int **PageSize** { get; set; }

Property value

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.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

Precision property

Specifies the floating-point precision used for string conversions by the new database.

Syntax

Visual Basic
Public Property **Precision** As Integer

C#
public int **Precision** { get; set; }

Property value

An integer specifying the precision. The value must be in the range [1,127]. The default is 30.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210
- "Scale property" on page 216

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.

Syntax

Visual Basic
Public Property **Scale** As Integer

C#
public int **Scale** { get; set; }

Property value

An integer specifying the scale. The value must be in the range [0,127]. The default is 6.

Remarks

Scale must be less than or equal to the Precision. If Scale is greater than the Precision, an error will occur while creating the database.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

TimeFormat property

Specifies the time format used for string conversions by the new database.

Syntax

Visual Basic
Public Property **TimeFormat** As String

C#
public string **TimeFormat** { get; set; }

Property value

A string specifying the time format. If the value is a null reference (Nothing in Visual Basic), the database will use "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).

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

TimestampFormat property

Specifies the timestamp format used for string conversions by the new database.

Syntax

Visual Basic
Public Property **TimestampFormat** As String

C#
public string **TimestampFormat** { get; set; }

Property value

A string specifying the timestamp format. If the value is a null reference (Nothing in Visual Basic), the database will use "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).

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

TimestampIncrement property

Specifies the minimum difference between two unique timestamp values, in microseconds (1,000,000th of a second).

Syntax

Visual Basic
Public Property **TimestampIncrement** As Integer

C#
public int **TimestampIncrement** { get; set; }

Property value

An integer specifying the timestamp increment. The value must be in the range [1,60000000]. The default is 1.

See also

- "ULCreateParms class" on page 209
- "ULCreateParms members" on page 210

UTF8Encoding property

Specifies whether the new database should be using the UTF8 character set or the character set associated with the collation.

Syntax

Visual Basic
Public Property **UTF8Encoding** As Boolean

C#
public bool **UTF8Encoding** { get; set; }

Property value

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.

Remarks

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.

For databases used on Palm OS devices, you must leave UTF8Encoding false.

See also

- “ULCreateParms class” on page 209
- “ULCreateParms members” on page 210

ToString method

Returns the string representation of this instance.

Syntax

Visual Basic
Public Overrides Function **ToString()** As String

C#
public override string **ToString();**

Return value

The string representation of this instance as a semicolon-separated list keyword=value pairs.

See also

- “ULCreateParms class” on page 209
- “ULCreateParms members” on page 210

ULCursorSchema class

UL Ext.: Represents the schema of an UltraLite.NET cursor. This class is abstract and so cannot be instantiated.

Syntax

Visual Basic
MustInherit Public Class **ULCursorSchema**

C#
public abstract class **ULCursorSchema**

Remarks

This class is an abstract base class of the ULTableSchema class and the ULResultSetSchema class.

Note to 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

- "ULCursorSchema members" on page 220
- "ULTableSchema class" on page 542
- "ULResultSetSchema class" on page 425

ULCursorSchema members

Public properties

Member name	Description
"ColumnCount property" on page 221	Returns the number of columns in the cursor.
"IsOpen property" on page 222	Checks whether the cursor schema is currently open.
"Name property" on page 222	Returns the name of the cursor.

Public methods

Member name	Description
"GetColumnID method" on page 222	Returns the column ID of the named column.
"GetColumnName method" on page 223	Returns the name of the column identified by the specified column ID.

Member name	Description
"GetColumnPrecision method" on page 224	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnSQLName method" on page 225	Returns the name of the column identified by the specified column ID.
"GetColumnScale method" on page 226	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnSize method" on page 226	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
"GetColumnULDbType method" on page 227	Returns the UltraLite.NET data type of the column identified by the specified column ID.
"GetSchemaTable method" on page 227	Returns a System.Data.DataTable that describes the column schema of the ULDataReader.

See also

- "ULCursorSchema class" on page 220
- "ULTableSchema class" on page 542
- "ULResultSetSchema class" on page 425

ColumnCount property

Returns the number of columns in the cursor.

Syntax

Visual Basic
Public Readonly Property **ColumnCount** As Short

C#
public short **ColumnCount** { get;}

Property value

The number of columns in the cursor or 0 if the cursor schema is closed.

Remarks

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.

See also

- “ULCursorSchema class” on page 220
- “ULCursorSchema members” on page 220

IsOpen property

Checks whether the cursor schema is currently open.

Syntax

Visual Basic
Public Readonly Property **IsOpen** As Boolean

C#
public bool **IsOpen** { get;}

Property value

True if the cursor schema is currently open, false if the cursor schema is closed.

See also

- “ULCursorSchema class” on page 220
- “ULCursorSchema members” on page 220

Name property

Returns the name of the cursor.

Syntax

Visual Basic
Public Readonly Property **Name** As String

C#
public string **Name** { get;}

Property value

The name of the cursor as a string.

See also

- “ULCursorSchema class” on page 220
- “ULCursorSchema members” on page 220

GetColumnID method

Returns the column ID of the named column.

Syntax

Visual Basic
Public Function **GetColumnID(** _
  ByVal *name* As String _
) As Short

C#
public short **GetColumnID(**
  string *name*
);

Parameters

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

Return value

The column ID of the named column.

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

- "ULCursorSchema class" on page 220
- "ULCursorSchema members" on page 220
- "ColumnCount property" on page 221
- "ColumnCount property" on page 221

GetColumnName method

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

Syntax

Visual Basic
Public Function **GetColumnName(** _
  ByVal *columnID* As Integer _
) As String

C#
public string **GetColumnName(**
  int *columnID*
);

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Return value

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.

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

- "ULCursorSchema class" on page 220
- "ULCursorSchema members" on page 220
- "ColumnCount property" on page 221
- "ColumnCount property" on page 221

GetColumnPrecision method

Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).

Syntax

Visual Basic

```
Public Function GetColumnPrecision( _
  ByVal columnID As Integer _
) As Integer
```

C#

```
public int GetColumnPrecision(
  int columnID
);
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Return value

The precision of the specified numeric column.

See also

- “ULCursorSchema class” on page 220
- “ULCursorSchema members” on page 220
- “GetColumnULDbType method” on page 227
- “ColumnCount property” on page 221

GetColumnSQLName method

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

Syntax

Visual Basic
Public Function **GetColumnSQLName(** _
  ByVal *columnID* As Integer _
) As String

C#
public string **GetColumnSQLName(**
  int *columnID*
);

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Return value

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.

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

- “ULCursorSchema class” on page 220
- “ULCursorSchema members” on page 220
- “ColumnCount property” on page 221
- “GetColumnName method” on page 223
- “ColumnCount property” on page 221

GetColumnScale method

Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).

Syntax

Visual Basic

Public Function **GetColumnScale(** _
  ByVal *columnID* As Integer _
) As Integer

C#

public int **GetColumnScale(**
  int *columnID*
);

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Return value

The scale of the specified numeric column.

See also

- "ULCursorSchema class" on page 220
- "ULCursorSchema members" on page 220
- "GetColumnULDbType method" on page 227
- "ColumnCount property" on page 221

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).

Syntax

Visual Basic

Public Function **GetColumnSize(** _
  ByVal *columnID* As Integer _
) As Integer

C#

public int **GetColumnSize(**
  int *columnID*
);

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Return value

The size of the specified sized column.

See also

- "ULCursorSchema class" on page 220
- "ULCursorSchema members" on page 220
- "GetColumnULDbType method" on page 227
- "ColumnCount property" on page 221

GetColumnULDbType method

Returns the UltraLite.NET data type of the column identified by the specified column ID.

Syntax

Visual Basic
Public Function **GetColumnULDbType(** _
  ByVal *columnID* As Integer _
) As ULDbType

C#
public ULDbType **GetColumnULDbType(**
  int *columnID*
);

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount-1].

Return value

A ULDbType enumerated integer.

See also

- "ULCursorSchema class" on page 220
- "ULCursorSchema members" on page 220
- "ColumnCount property" on page 221
- "ColumnCount property" on page 221
- "ULDbType enumeration" on page 301

GetSchemaTable method

Returns a System.Data.DataTable that describes the column schema of the ULDataReader.

Syntax

Visual Basic
Public Function **GetSchemaTable()** As DataTable

C#
public DataTable **GetSchemaTable();**

Return value

A System.Data.DataTable that describes the column schema.

Remarks

For more information, see “GetSchemaTable method” on page 288.

See also

- “ULCursorSchema class” on page 220
- “ULCursorSchema members” on page 220
- DataTable
- “ULDataReader class” on page 261

ULDataAdapter class

Represents a set of commands and a database connection used to fill a System.Data.DataSet and to update a database. This class cannot be inherited.

Syntax

Visual Basic

```
Public NotInheritable Class ULDataAdapter
  Inherits DbDataAdapter
```

C#

```
public sealed class ULDataAdapter: DbDataAdapter
```

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.

Inherits: System.Data.Common.DbDataAdapter

Implements: System.Data.IDbDataAdapter, System.Data.IDataAdapter, System.IDisposable

See also

- "ULDataAdapter members" on page 229
- DataSet
- "ULDataReader class" on page 261
- "ULTable class" on page 520
- DbDataAdapter
- IDbDataAdapter
- IDataAdapter
- IDisposable

ULDataAdapter members

Public constructors

Member name	Description
"ULDataAdapter constructors" on page 232	Initializes a new instance of the "ULDataAdapter class" on page 229.

Public properties

Member name	Description
AcceptChangesDuringFill (inherited from DataAdapter)	Gets or sets a value indicating whether DataRow.AcceptChanges is called on a DataRow after it is added to the DataTable during any of the Fill operations.
AcceptChangesDuringUpdate (inherited from DataAdapter)	Gets or sets whether DataRow.AcceptChanges is called during a DataAdapter.Update.
ContinueUpdateOnError (inherited from DataAdapter)	Gets or sets a value that specifies whether to generate an exception when an error is encountered during a row update.
"DeleteCommand property" on page 235	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.
FillLoadOption (inherited from DataAdapter)	Gets or sets the LoadOption that determines how the adapter fills the DataTable from the DbDataReader.
"InsertCommand property" on page 235	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.
MissingMappingAction (inherited from DataAdapter)	Determines the action to take when incoming data does not have a matching table or column.
MissingSchemaAction (inherited from DataAdapter)	Determines the action to take when existing DataSet schema does not match incoming data.
ReturnProviderSpecificTypes (inherited from DataAdapter)	Gets or sets whether the DataAdapter.Fill should return provider-specific values or common CLS-compliant values.
"SelectCommand property" on page 236	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.
"TableMappings property" on page 237	Returns a collection that provides the master mapping between a source table and a System.Data.DataTable
UpdateBatchSize (inherited from 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.

Member name	Description
"UpdateCommand property" on page 238	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.

Public methods

Member name	Description
Fill (inherited from DbDataAdapter)	Fills a DataSet or a DataTable.
FillSchema (inherited from DbDataAdapter)	Adds a DataTable to a DataSet and configures the schema to match that in the data source.
"GetFillParameters method" on page 239	Returns the parameters set by the user when executing a SELECT statement.
ResetFillLoadOption (inherited from DataAdapter)	Resets DataAdapter.FillLoadOption to its default state and causes DataAdapter.Fill to honor DataAdapter.AcceptChangesDuringFill.
ShouldSerializeAcceptChangesDuringFill (inherited from DataAdapter)	Determines whether the DataAdapter.AcceptChangesDuringFill should be persisted.
ShouldSerializeFillLoadOption (inherited from DataAdapter)	Determines whether the DataAdapter.FillLoadOption should be persisted.
Update (inherited from DbDataAdapter)	Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the DataSet.

Public events

Member name	Description
FillError (inherited from DataAdapter)	Returned when an error occurs during a fill operation.
"RowUpdated event" on page 239	Occurs during an update after a command is executed against the data source. When an attempt to update is made, the event fires.
"RowUpdating event" on page 240	Occurs during an update before a command is executed against the data source. When an attempt to update is made, the event fires.

See also

- “ULDataAdapter class” on page 229
- DataSet
- “ULDataReader class” on page 261
- “ULTable class” on page 520
- DbDataAdapter
- IDbDataAdapter
- IDataAdapter
- IDisposable

ULDataAdapter constructors

Initializes a new instance of the “ULDataAdapter class” on page 229.

ULDataAdapter() constructor

Initializes a ULDataAdapter object.

Syntax

Visual Basic
Public Sub **New()**

C#
public **ULDataAdapter();**

See also

- “ULDataAdapter class” on page 229
- “ULDataAdapter members” on page 229
- “ULDataAdapter constructors” on page 232
- “ULDataAdapter(ULCommand) constructor” on page 232
- “ULDataAdapter(String, ULConnection) constructor” on page 233
- “ULDataAdapter(String, String) constructor” on page 234

ULDataAdapter(ULCommand) constructor

Initializes a ULDataAdapter object with the specified SELECT statement.

Syntax

Visual Basic
Public Sub **New(** _
  ByVal *selectCommand* As ULCommand _
)

C#
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 class" on page 229
- "ULDataAdapter members" on page 229
- "ULDataAdapter constructors" on page 232
- "ULDataAdapter() constructor" on page 232
- "ULDataAdapter(String, ULConnection) constructor" on page 233
- "ULDataAdapter(String, String) constructor" on page 234
- "ULCommand class" on page 86
- DbDataAdapter.Fill
- DataSet

ULDataAdapter(String, ULConnection) constructor

Initializes a ULDataAdapter object with the specified SELECT statement and connection.

Syntax

Visual Basic

```
Public Sub New( _
  ByVal selectCommandText As String, _
  ByVal selectConnection As ULConnection _
)
```

C#

```
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 class" on page 229
- "ULDataAdapter members" on page 229
- "ULDataAdapter constructors" on page 232
- "ULDataAdapter() constructor" on page 232
- "ULDataAdapter(ULCommand) constructor" on page 232
- "ULDataAdapter(String, String) constructor" on page 234
- "SelectCommand property" on page 236
- "ULConnection class" on page 131

ULDataAdapter(String, String) constructor

Initializes a ULDataAdapter object with the specified SELECT statement and connection string.

Syntax

Visual Basic

```
Public Sub New( _
   ByVal selectCommandText As String, _
   ByVal selectConnectionString As String _
)
```

C#

```
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 class" on page 229
- "ULDataAdapter members" on page 229
- "ULDataAdapter constructors" on page 232
- "ULDataAdapter() constructor" on page 232
- "ULDataAdapter(ULCommand) constructor" on page 232
- "ULDataAdapter(String, ULConnection) constructor" on page 233
- "SelectCommand property" on page 236

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.

Syntax

Visual Basic
Public Property **DeleteCommand** As ULCommand

C#
public ULCommand **DeleteCommand** { get; set; }

Property value

A ULCommand object that is executed to delete rows in the database that correspond to deleted rows in the System.Data.DataSet.

Remarks

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

- "ULDataAdapter class" on page 229
- "ULDataAdapter members" on page 229
- "ULCommand class" on page 86
- DbDataAdapter.Update
- DataSet
- "ULCommand class" on page 86
- IDbDataAdapter.DeleteCommand
- DbDataAdapter.DeleteCommand

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.

Syntax

Visual Basic
Public Property **InsertCommand** As ULCommand

C#
public ULCommand **InsertCommand** { get; set; }

Property value

A ULCommand object that is executed to insert rows in the database that correspond to inserted rows in the System.Data.DataSet.

Remarks

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

- "ULDataAdapter class" on page 229
- "ULDataAdapter members" on page 229
- "ULCommand class" on page 86
- DbDataAdapter.Update
- DataSet
- "ULCommand class" on page 86
- IDbDataAdapter.InsertCommand
- DbDataAdapter.InsertCommand

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.

Syntax

Visual Basic
Public Property **SelectCommand** As ULCommand

C#
public ULCommand **SelectCommand** { get; set; }

Property value

A ULCommand object that is executed to fill the System.Data.DataSet.

Remarks

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

- "ULDataAdapter class" on page 229
- "ULDataAdapter members" on page 229
- "ULCommand class" on page 86
- DataSet
- DbDataAdapter.Fill
- DbDataAdapter.FillSchema
- "ULDataAdapter(ULCommand) constructor" on page 232
- "ULDataAdapter(String, ULConnection) constructor" on page 233
- "ULDataAdapter(String, String) constructor" on page 234
- IDbDataAdapter.SelectCommand
- DbDataAdapter.SelectCommand

TableMappings property

Returns a collection that provides the master mapping between a source table and a System.Data.DataTable

Syntax

Visual Basic
Public Readonly Property **TableMappings** As DataTableMappingCollection

C#
public DataTableMappingCollection **TableMappings** { get;}

Property value

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.

Remarks

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

- “ULDataAdapter class” on page 229
- “ULDataAdapter members” on page 229
- DataTable
- DataTableMapping
- DataTable
- DataTableMappingCollection
- DataSet
- IDataAdapter.TableMappings

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.

Syntax

Visual Basic
Public Property **UpdateCommand** As ULCommand

C#
public ULCommand **UpdateCommand** { get; set; }

Property value

A ULCommand object that is executed to update rows in the database that correspond to updated rows in the System.Data.DataSet.

Remarks

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

- “ULDataAdapter class” on page 229
- “ULDataAdapter members” on page 229
- “ULCommand class” on page 86
- DataSet
- DbDataAdapter.Update
- “UpdatedRowSource property” on page 99
- IDbDataAdapter.UpdateCommand
- DbDataAdapter.DeleteCommand

GetFillParameters method

Returns the parameters set by the user when executing a SELECT statement.

Syntax

Visual Basic
Public Function **GetFillParameters()** As ULParameter

C#
public ULParameter **GetFillParameters();**

Return value

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

- "ULDataAdapter class" on page 229
- "ULDataAdapter members" on page 229
- DbDataAdapter.GetFillParameters
- "ULParameter class" on page 358

RowUpdated event

Occurs during an update after a command is executed against the data source. When an attempt to update is made, the event fires.

Syntax

Visual Basic
Public Event **RowUpdated** As ULRowUpdatedEventHandler

C#
public event ULRowUpdatedEventHandler **RowUpdated**;

Remarks

To process row updated events, you must create a ULRowUpdatedEventHandler delegate and attach it to this event.

Event data

- **Command** Returns the ULCommand executed when DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping) is called.
- **RecordsAffected** Returns the number of rows changed, inserted, or deleted by the execution of the SQL statement. For SELECT statements this value is -1.
- **Command** Gets the IDbCommand executed when DbDataAdapter.Update is called.

- **Errors** Gets any errors generated by the .NET Framework data provider when the RowUpdatedEventArgs.Command was executed.
- **Row** Gets the DataRow sent through an DbDataAdapter.Update.
- **RowCount** Gets the number of rows processed in a batch of updated records.
- **StatementType** Gets the type of SQL statement executed.
- **Status** Gets the UpdateStatus of the RowUpdatedEventArgs.Command.
- **TableMapping** Gets the DataTableMapping sent through an DbDataAdapter.Update.

See also

- "ULDataAdapter class" on page 229
- "ULDataAdapter members" on page 229
- "ULRowUpdatedEventHandler delegate" on page 436

RowUpdating event

Occurs during an update before a command is executed against the data source. When an attempt to update is made, the event fires.

Syntax

Visual Basic
Public Event **RowUpdating** As ULRowUpdatingEventHandler

C#
public event ULRowUpdatingEventHandler **RowUpdating**;

Remarks

To process row updating events, you must create a ULRowUpdatingEventHandler delegate and attach it to this event.

Event data

- **Command** Specifies the ULCommand to execute when performing the DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping).
- **Command** Gets the IDbCommand to execute during the DbDataAdapter.Update operation.
- **Errors** Gets any errors generated by the .NET Framework data provider when the RowUpdatedEventArgs.Command executes.
- **Row** Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
- **StatementType** Gets the type of SQL statement to execute.
- **Status** Gets or sets the UpdateStatus of the RowUpdatedEventArgs.Command.
- **TableMapping** Gets the DataTableMapping to send through the DbDataAdapter.Update.

See also

- "ULDataAdapter class" on page 229
- "ULDataAdapter members" on page 229
- "ULRowUpdatedEventHandler delegate" on page 436

ULDatabaseManager class

UL Ext.: Manages synchronization listeners and the UltraLite.NET runtime type. The ULDatabaseManager class also allows you to drop (delete) UltraLite.NET databases. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULDatabaseManager**

C#
public sealed class **ULDatabaseManager**

Remarks

This class is a singleton class whose only instance is accessible through the static (Shared in Visual Basic) ULConnection.DatabaseManager.

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

// C#
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT;
ULConnection conn = new ULConnection();
// The RuntimeType is now locked
```

See also

- “ULDatabaseManager members” on page 242
- “DatabaseManager property” on page 140
- “RuntimeType property” on page 243

ULDatabaseManager members

Public properties

Member name	Description
“RuntimeType property” on page 243	Specifies the UltraLite.NET runtime type. The runtime type must be selected before using any other UltraLite.NET API.

Public methods

Member name	Description
"CreateDatabase method" on page 244	Creates a new UltraLite database.
"DropDatabase method" on page 245	Deletes the specified database. You cannot drop a database that has open connections.
"SetActiveSyncListener method" on page 246	Specifies the listener object used to process ActiveSync calls from the MobiLink provider for ActiveSync.
"SetGlobalListener method" on page 247	Specifies the listener objects used to process global synchronization and SQL pass-through messages.
"SetServerSyncListener method" on page 248	Specifies the listener object used to process the specified server synchronization message.
"SignalSyncIsComplete method" on page 249	Signals the MobiLink provider for ActiveSync that an application has completed synchronization.
"ValidateDatabase method" on page 250	Performs low level and index validation on a database.

See also

- "ULDatabaseManager class" on page 242
- "DatabaseManager property" on page 140
- "RuntimeType property" on page 243

RuntimeType property

Specifies the UltraLite.NET runtime type. The runtime type must be selected before using any other UltraLite.NET API.

Syntax

Visual Basic
Public Shared Property **RuntimeType** As ULRuntimeType

C#
public static ULRuntimeType **RuntimeType** { get; set; }

Property value

A ULRuntimeType value identifying the type of the unmanaged UltraLite .NET runtime.

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

// C#
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT;
ULConnection conn = new ULConnection();
// The RuntimeType is now locked
```

See also

- "ULDatabaseManager class" on page 242
- "ULDatabaseManager members" on page 242
- "ULRuntimeType enumeration" on page 441

CreateDatabase method

Creates a new UltraLite database.

Syntax

Visual Basic
Public Sub **CreateDatabase(** _
  ByVal *connString* As String, _
  ByVal *collationData* As Byte(), _
  ByVal *createParms* As String _
)

C#
public void **CreateDatabase(**
  string *connString*,
  byte[] *collationData*,
  string *createParms*
);

Parameters

- **connString** The parameters for identifying a database in the form of a semicolon-separated list of keyword-value pairs. For more information, see "ULConnectionParms class" on page 179.
- **collationData** The collation data specifying how the database will store and compare strings.
- **createParms** The parameters used to configure the new database in the form of a semicolon-separated list of keyword-value pairs. For more information, see "ULCreateParms class" on page 209.

Remarks

To specify a collation, you must first include the appropriate collation data source file from the UltraLite \Collations\cs (for C# projects) or UltraLite\Collations\vb.net (for Visual Basic projects) subdirectory of your SQL Anywhere installation directory. Once included in your project, use the collation's Data property to supply the collation data to the CreateDatabase() method.

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.DatabaseOnCE = "\UltraLite\MyDatabase.udb"
' Assumes file UltraLite\Collations\vb.net\coll_1250LATIN2.vb is
' also compiled in the current project
ULConnection.DatabaseManager.CreateDatabase( _
    openParms.ToString(), _
    iAnywhere.UltraLite.Collations.Collation_1250LATIN2.Data, _
    "" _
  )
Dim conn As ULConnection = _
  New ULConnection( openParms.ToString() )
conn.Open()
```

```
// C#
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnCE = @"\UltraLite\MyDatabase.udb";
// Assumes file UltraLite\Collations\cs\coll_1250LATIN2.cs is
// also compiled in the current project
ULConnection.DatabaseManager.CreateDatabase(
    openParms.ToString(),
    iAnywhere.UltraLite.Collations.Collation_1250LATIN2.Data,
    ""
  );
ULConnection conn = new ULConnection( openParms.ToString() );
conn.Open();
```

See also

- "ULDatabaseManager class" on page 242
- "ULDatabaseManager members" on page 242
- "Open method" on page 167

DropDatabase method

Deletes the specified database.

You cannot drop a database that has open connections.

Syntax

Visual Basic
Public Sub **DropDatabase(** _
  ByVal *connString* As String _
)

C#
public void **DropDatabase(**

string *connString*
);

Parameters

- **connString** The parameters for identifying a database in the form of a semicolon-separated list of keyword-value pairs. For more information, see "ULConnectionParms 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 connParms As ULConnectionParms = New ULConnectionParms
connParms.DatabaseOnCE = "\UltraLite\MyDatabase.udb"
ULConnection.DatabaseManager.DropDatabase( _
    connParms.ToString() _
  )


// C#
ULConnectionParms connParms = new ULConnectionParms();
connParms.DatabaseOnCE = @"\UltraLite\MyDatabase.udb";
ULConnection.DatabaseManager.DropDatabase(
    connParms.ToString()
  );
ULConnection conn = new ULConnection( openParms.ToString() );
conn.Open();
```

See also

- "ULDatabaseManager class" on page 242
- "ULDatabaseManager members" on page 242
- "Open method" on page 167

SetActiveSyncListener method

Specifies the listener object used to process ActiveSync calls from the MobiLink provider for ActiveSync.

Syntax

Visual Basic
Public Sub **SetActiveSyncListener(** _
  ByVal *appClassName* As String, _
  ByVal *listener* As ULActiveSyncListener _
)

C#
public 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.

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.

Example

For an example of SetActiveSyncListener, see "ActiveSyncInvoked method" on page 53.

See also

- "ULDatabaseManager class" on page 242
- "ULDatabaseManager members" on page 242
- "SetServerSyncListener method" on page 248
- "SetActiveSyncListener method" on page 246
- "ULActiveSyncListener interface" on page 53
- "ActiveSyncInvoked method" on page 53

SetGlobalListener method

Specifies the listener objects used to process global synchronization and SQL pass-through messages.

Syntax

Visual Basic

```
Public Sub SetGlobalListener( _
  ByVal syncListener As ULSyncProgressListener, _
  ByVal sqlListener As ULSqlPassthroughProgressListener _
)
```

C#

```
public void SetGlobalListener(
  ULSyncProgressListener syncListener,
  ULSqlPassthroughProgressListener sqlListener
);
```

Parameters

- **syncListener** The ULSyncProgressListener object that implements SyncProgressed(), which is called for global synchronization messages.
- **sqlListener** The ULSqlPassthroughProgressListener object that implements ScriptProgressed(), which is called as each SQL pass-through script is executed.

Remarks

When the SYNCHRONIZE profileName SQL statement is executed, its progress messages are routed to syncListener, if not null (Nothing in Visual Basic).

Several SQL scripts may be available and automatically executed when a database is connected. Also, scripts may be passed down in a subsequent synchronization, and executed directly with ULConnection.ExecuteSQLPassthroughScripts(). In either case, script progress messages will be routed to sqlListener, if not null (Nothing in Visual Basic).

To remove either listener pass a null reference in a call to SetGlobalListener. As of 11.0, applications no longer need to remove listeners before exiting.

See also

- “ULDatabaseManager class” on page 242
- “ULDatabaseManager members” on page 242
- “ULSyncProgressListener interface” on page 510
- “ULSqlPassthroughProgressListener interface” on page 457
- “ExecuteSQLPassthroughScripts method” on page 155

SetServerSyncListener method

Specifies the listener object used to process the specified server synchronization message.

Syntax

Visual Basic

```
Public Sub SetServerSyncListener( _
  ByVal messageName As String, _
  ByVal appClassName As String, _
  ByVal listener As ULServerSyncListener _
)
```

C#

```
public 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.

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.

Example

For an example of SetServerSyncListener, see “ServerSyncInvoked method” on page 442.

See also

- “ULDatabaseManager class” on page 242
- “ULDatabaseManager members” on page 242
- “SetServerSyncListener method” on page 248
- “SetActiveSyncListener method” on page 246
- “ServerSyncInvoked method” on page 442

SignalSyncIsComplete method

Signals the MobiLink provider for ActiveSync that an application has completed synchronization.

Syntax

Visual Basic
Public Sub **SignalSyncIsComplete()**

C#
public void **SignalSyncIsComplete();**

Example

For an example of SignalSyncIsComplete, see “ActiveSyncInvoked method” on page 53.

See also

- “ULDatabaseManager class” on page 242
- “ULDatabaseManager members” on page 242
- “SignalSyncIsComplete method” on page 249

ValidateDatabase method

Performs low level and index validation on a database.

Syntax

Visual Basic
Public Sub **ValidateDatabase(** _
  ByVal *start_parms* As String, _
  ByVal *how* As ULDBValid _
)

C#
public 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. For more information, see "ULConnectionParms class" on page 179.
- **how** How to validate the database. For more information, see "ULDBValid enumeration" on page 305.

Example

The following code validates indexes for the database \UltraLite\MyDatabase.udb under Windows Mobile

```
' Visual Basic
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnCE = "\UltraLite\MyDatabase.udb"
ULConnection.DatabaseManager.ValidateDatabase( _
    openParms.ToString(), iAnywhere.Data.UltraLite.ULVF_INDEX )
```

```
// C#
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnCE = @"\UltraLite\MyDatabase.udb";
ULConnection.DatabaseManager.ValidateDatabase(
    openParms.ToString(), iAnywhere.Data.UltraLite.ULVF_INDEX );
```

See also

- "ULDatabaseManager class" on page 242
- "ULDatabaseManager members" on page 242
- "ValidateDatabase method" on page 175
- "ULDBValid enumeration" on page 305

ULDatabaseSchema class

UL Ext.: Represents the schema of an UltraLite.NET database. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULDatabaseSchema**

C#
public sealed class **ULDatabaseSchema**

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 members" on page 251
- "ULDatabaseSchema class" on page 251
- "Schema property" on page 142

ULDatabaseSchema members

Public properties

Member name	Description
"CollationName property" on page 252	This property has been deprecated. Use GetDatabaseProperty("Collation") instead. The name of the database's collation sequence.
"IsCaseSensitive property" on page 253	Checks whether the database is case sensitive.
"IsOpen property" on page 253	Whether the database schema is open.
"PublicationCount property" on page 254	The number of publications in the database.
"TableCount property" on page 254	The number of tables in the database.

Public methods

Member name	Description
"GetDatabaseProperty method" on page 255	Returns the value of the specified database property.

Member name	Description
"GetPublicationName method" on page 257	Returns the name of the publication identified by the specified publication ID.
"GetPublicationSchema method" on page 258	Returns the publication schema corresponding to the named publication.
"GetTableName method" on page 258	Returns the name of the table identified by the specified table ID.
"SetDatabaseOption method" on page 259	Sets the value for the specified database option.

See also

- "ULDatabaseSchema class" on page 251
- "ULDatabaseSchema class" on page 251
- "Schema property" on page 142

CollationName property

This property has been deprecated. Use GetDatabaseProperty("Collation") instead. The name of the database's collation sequence.

Syntax

Visual Basic
Public Readonly Property **CollationName** As String

C#
public string **CollationName** { get;}

Property value

A string representing the database's collation sequence.

Remarks

The database collation sequence affects how indexes on tables and result sets are sorted.

See also

- "ULDatabaseSchema class" on page 251
- "ULDatabaseSchema members" on page 251
- "GetDatabaseProperty method" on page 255

IsCaseSensitive property

Checks whether the database is case sensitive.

Syntax

Visual Basic
Public Readonly Property **IsCaseSensitive** As Boolean

C#
public bool **IsCaseSensitive** { get;}

Property value

True if the database is case sensitive, and false if the database is case insensitive.

Remarks

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

- "ULDatabaseSchema class" on page 251
- "ULDatabaseSchema members" on page 251
- "GetDatabaseProperty method" on page 255
- "UserID property" on page 189
- "Password property" on page 188

IsOpen property

Whether the database schema is open.

Syntax

Visual Basic
Public Readonly Property **IsOpen** As Boolean

C#
public bool **IsOpen** { get;}

Property value

True if this database schema is currently open, false if this database schema is currently closed.

Remarks

A ULDatabaseSchema object is open only if the connection it is attached to is open.

See also

- "ULDatabaseSchema class" on page 251
- "ULDatabaseSchema members" on page 251

PublicationCount property

The number of publications in the database.

Syntax

Visual Basic
Public Readonly Property **PublicationCount** As Integer

C#
public int **PublicationCount** { get;}

Property value

The number of publications in the database or zero if the connection is not open.

Remarks

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

- "ULDatabaseSchema class" on page 251
- "ULDatabaseSchema members" on page 251
- "GetPublicationName method" on page 257

TableCount property

The number of tables in the database.

Syntax

Visual Basic
Public Readonly Property **TableCount** As Integer

C#
public int **TableCount** { get;}

Property value

The number of tables in the database or zero if the connection is not open.

Remarks

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.

See also

- “ULDatabaseSchema class” on page 251
- “ULDatabaseSchema members” on page 251

GetDatabaseProperty method

Returns the value of the specified database property.

Syntax

Visual Basic

```
Public Function GetDatabaseProperty( _
  ByVal name As String _
) As String
```

C#

```
public string  GetDatabaseProperty(
  string  name
);
```

Parameters

- **name** The name of the database property whose value you want to obtain. Property names are case insensitive.

Return value

The value of the property as a string.

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.

Property	Description
CollationName	This property has been deprecated. Use "Collation" instead.
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).
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.

Property	Description
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 timestamp values, in microseconds (1,000,000th of a second).

See also

- "ULDatabaseSchema class" on page 251
- "ULDatabaseSchema members" on page 251
- "SetDatabaseOption method" on page 259
- "CollationName property" on page 252
- "UserID property" on page 189
- "Password property" on page 188
- TimeSpan
- DateTime

GetPublicationName method

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

Syntax

Visual Basic

```
Public Function GetPublicationName( _
  ByVal pubID As Integer _
) As String
```

C#

```
public string  GetPublicationName(
  int pubID
);
```

Parameters

- **pubID** The ID of the publication. The value must be in the range [1,PublicationCount].

Return value

The publication name as a string.

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

- “ULDatabaseSchema class” on page 251
- “ULDatabaseSchema members” on page 251
- “PublicationCount property” on page 254
- “PublicationCount property” on page 254

GetPublicationSchema method

Returns the publication schema corresponding to the named publication.

Syntax

Visual Basic

Public Function **GetPublicationSchema(** _
  ByVal *name* As String _
) As ULPublicationSchema

C#

public ULPublicationSchema **GetPublicationSchema(**
  string *name*
);

Parameters

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

Return value

The ULPublicationSchema object representing the named publication.

See also

- “ULDatabaseSchema class” on page 251
- “ULDatabaseSchema members” on page 251
- “GetPublicationName method” on page 257
- “ULPublicationSchema class” on page 395

GetTableName method

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

Syntax

Visual Basic

Public Function **GetTableName(** _
  ByVal *tableID* As Integer _
) As String

C#

public string **GetTableName(**

```
  int tableID
);
```

Parameters

- **tableID** The ID of the table. The value must be in range [1,TableCount].

Return value

The table name as a string.

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

- "ULDatabaseSchema class" on page 251
- "ULDatabaseSchema members" on page 251
- "TableCount property" on page 254

SetDatabaseOption method

Sets the value for the specified database option.

Syntax

Visual Basic

```
Public Sub SetDatabaseOption( _
  ByVal name As String, _
  ByVal value As String _
)
```

C#

```
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.

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

- "ULDatabaseSchema class" on page 251
- "ULDatabaseSchema members" on page 251
- "GetDatabaseProperty method" on page 255
- UInt32.MaxValue
- "INVALID_DATABASE_ID field" on page 137
- "BeginTransaction(IsolationLevel) method" on page 146

ULDataReader class

Represents a read-only bi-directional cursor in an UltraLite database. Cursors are sets of rows from either a table or the result set from a query.

Syntax

Visual Basic
Public Class **ULDataReader**
  Inherits DbDataReader

C#
public class **ULDataReader**: DbDataReader

Remarks

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 FOR READ ONLY", conn _
  )
Dim reader As ULDataReader = cmd.ExecuteReader()

// C#
ULCommand cmd = new ULCommand(
    "SELECT emp_id FROM employee FOR READ ONLY", 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.

Inherits: System.Data.Common.DbDataReader

Implements: System.Data.IDataReader, System.Data.IDataRecord, System.IDisposable, System.ComponentModel.IListSource

See also

- "ULDataReader members" on page 262
- "ULCommand class" on page 86
- "ExecuteResultSet() method" on page 115
- "ExecuteTable() method" on page 118
- "ULDataAdapter class" on page 229
- "GetString method" on page 290
- DbDataReader
- IDataReader
- IDataRecord
- IDisposable
- IListSource

ULDataReader members

Public properties

Member name	Description
"Depth property" on page 266	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
"FieldCount property" on page 266	Returns the number of columns in the cursor.
"HasRows property" on page 267	Checks whether the ULDataReader has one or more rows.
"IsBOF property" on page 267	**UL Ext.:** Checks whether the current row position is before the first row.
"IsClosed property" on page 268	Checks whether the cursor is currently open.
"IsEOF property" on page 268	**UL Ext.:** Checks whether the current row position is after the last row.
"Item properties" on page 268	Gets the value of a specified column as an instance of Object.
"RecordsAffected property" on page 270	Returns the number of rows changed, inserted, or deleted by execution of the SQL statement. For SELECT statements or CommandType.TableDirect tables, this value is -1.
"RowCount property" on page 271	**UL Ext.:** Returns the number of rows in the cursor.
"Schema property" on page 271	**UL Ext.:** Holds the schema of this cursor.

Member name	Description
VisibleFieldCount (inherited from DbDataReader)	Gets the number of fields in the DbDataReader that are not hidden.

Public methods

Member name	Description
"Close method" on page 272	Closes the cursor.
Dispose (inherited from DbDataReader)	Releases the resources consumed by this DbDataReader.
"GetBoolean method" on page 272	Returns the value for the specified column as a System.Boolean.
"GetByte method" on page 273	Returns the value for the specified column as an unsigned 8-bit value (System.Byte).
"GetBytes methods" on page 273	**UL Ext.:** Returns the value for the specified column as an array of System.Bytes. Only valid for columns of type ULDbType.Binary, ULDbType.LongBinary, or ULDbType.UniqueIdentifier.
"GetChar method" on page 276	This method is not supported in UltraLite.NET.
"GetChars method" on page 277	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 (inherited from DbDataReader)	Returns a DbDataReader object for the requested column ordinal.
"GetDataTypeName method" on page 278	Returns the name of the specified column's provider data type.
"GetDateTime method" on page 279	Returns the value for the specified column as a System.DateTime with millisecond accuracy.
"GetDecimal method" on page 279	Returns the value for the specified column as a System.Decimal.
"GetDouble method" on page 280	Returns the value for the specified column as a System.Double.
"GetEnumerator method" on page 281	Returns an System.Collections.IEnumerator that iterates through the ULDataReader.

Member name	Description
"GetFieldType method" on page 281	Returns the System.Type most appropriate for the specified column.
"GetFloat method" on page 282	Returns the value for the specified column as a System.Single.
"GetGuid method" on page 282	Returns the value for the specified column as a UUID (System.Guid).
"GetInt16 method" on page 283	Returns the value for the specified column as a System.Int16.
"GetInt32 method" on page 284	Returns the value for the specified column as an Int32.
"GetInt64 method" on page 285	Returns the value for the specified column as an Int64.
"GetName method" on page 285	Returns the name of the specified column.
"GetOrdinal method" on page 286	Returns the column ID of the named column.
GetProviderSpecificFieldType (inherited from DbDataReader)	Returns the provider-specific field type of the specified column.
GetProviderSpecificValue (inherited from DbDataReader)	Gets the value of the specified column as an instance of Object.
GetProviderSpecificValues (inherited from DbDataReader)	Gets all provider-specific attribute columns in the collection for the current row.
"GetSchemaTable method" on page 288	Returns a System.Data.DataTable that describes the column metadata of the ULDataReader.
"GetString method" on page 290	Returns the value for the specified column as a System.String.
"GetTimeSpan method" on page 290	Returns the value for the specified column as a System.TimeSpan with millisecond accuracy.
"GetUInt16 method" on page 291	Returns the value for the specified column as a System.UInt16.
"GetUInt32 method" on page 292	Returns the value for the specified column as a UInt32.
"GetUInt64 method" on page 292	Returns the value for the specified column as a System.UInt64.

Member name	Description
"GetValue method" on page 293	Returns the value of the specified column in its native format.
"GetValues method" on page 294	Returns all the column values for the current row.
"IsDBNull method" on page 295	Checks whether the value from the specified column is NULL.
"MoveAfterLast method" on page 295	**UL Ext.:** Positions the cursor to after the last row of the cursor.
"MoveBeforeFirst method" on page 296	**UL Ext.:** Positions the cursor to before the first row of the cursor.
"MoveFirst method" on page 296	**UL Ext.:** Positions the cursor to the first row of the cursor.
"MoveLast method" on page 296	**UL Ext.:** Positions the cursor to the last row of the cursor.
"MoveNext method" on page 297	**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" on page 297	**UL Ext.:** Positions the cursor to the previous row or before the first row.
"MoveRelative method" on page 298	**UL Ext.:** Positions the cursor relative to the current row.
"NextResult method" on page 299	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
"Read method" on page 299	Positions the cursor to the next row, or after the last row if the cursor was already on the last row.

See also

- "ULDataReader class" on page 261
- "ULCommand class" on page 86
- "ExecuteResultSet() method" on page 115
- "ExecuteTable() method" on page 118
- "ULDataAdapter class" on page 229
- "GetString method" on page 290
- DbDataReader
- IDataReader
- IDataRecord
- IDisposable

Depth property

Returns the depth of nesting for the current row. The outermost table has a depth of zero.

Syntax

Visual Basic
Public Overrides Readonly Property **Depth** As Integer

C#
public override int **Depth** { get;}

Property value

All UltraLite.NET result sets have a depth of zero.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

FieldCount property

Returns the number of columns in the cursor.

Syntax

Visual Basic
Public Overrides Readonly Property **FieldCount** As Integer

C#
public override int **FieldCount** { get;}

Property value

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

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "ColumnCount property" on page 221

HasRows property

Checks whether the ULDataReader has one or more rows.

Syntax

Visual Basic
Public Overrides Readonly Property **HasRows** As Boolean

C#
public override bool **HasRows** { get;}

Property value

True if the result set has at least one row, false if there are no rows.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

IsBOF property

UL Ext.: Checks whether the current row position is before the first row.

Syntax

Visual Basic
Public Readonly Property **IsBOF** As Boolean

C#
public bool **IsBOF** { get;}

Property value

True if the current row position is before the first row, false otherwise.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

IsClosed property

Checks whether the cursor is currently open.

Syntax

Visual Basic
Public Overrides Readonly Property **IsClosed** As Boolean

C#
public override bool **IsClosed** { get;}

Property value

True if the cursor is currently open, false if the cursor is closed.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

IsEOF property

UL Ext.: Checks whether the current row position is after the last row.

Syntax

Visual Basic
Public Readonly Property **IsEOF** As Boolean

C#
public bool **IsEOF** { get;}

Property value

True if the current row position is after the last row, false otherwise.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

Item properties

Gets the value of a specified column as an instance of Object.

Item(Int32) property

Returns the value of the specified column in its native format. In C#, this property is the indexer for the ULDataReader class.

Syntax

Visual Basic
Public Overrides Default Readonly Property **Item(** _
  ByVal *columnID* As Integer _
) As Object

C#
public override object **this[**
  int *columnID*
] { get;}

Parameters

- **columnID** 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.

Property value

The column value as the .NET type most appropriate for the column or DBNull if column is NULL.

Remarks

This method is identical in functionality to the ULDataReader.GetValue(int) method.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "Item properties" on page 268
- "GetFieldType method" on page 281
- "Item(String) property" on page 269
- "GetValue method" on page 293
- "FieldCount property" on page 266

Item(String) property

Returns the value of the specified named column in its native format. In C#, this property is the indexer for the ULDataReader class.

Syntax

Visual Basic
Public Overrides Default Readonly Property **Item(** _
  ByVal *name* As String _
) As Object

C#
public override object **this[**
  string *name*
] { get;}

Parameters

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

Property value

The column value as the .NET type most appropriate for the column or DBNull if column is NULL.

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".

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

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "Item properties" on page 268
- "Item(Int32) property" on page 268
- "GetOrdinal method" on page 286
- "GetValue method" on page 293
- "GetFieldType method" on page 281

RecordsAffected property

Returns the number of rows changed, inserted, or deleted by execution of the SQL statement. For SELECT statements or CommandType.TableDirect tables, this value is -1.

Syntax

Visual Basic
Public Overrides Readonly Property **RecordsAffected** As Integer

C#
public override int **RecordsAffected** { get;}

Property value

The number of rows changed, inserted, or deleted by execution of the SQL statement.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- CommandType.TableDirect

RowCount property

UL Ext.: Returns the number of rows in the cursor.

Syntax

Visual Basic
Public Readonly Property **RowCount** As Integer

C#
public int **RowCount** { get;}

Property value

The number of rows in the cursor.

Remarks

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

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "StartSynchronizationDelete method" on page 171
- "StopSynchronizationDelete method" on page 172

Schema property

UL Ext.: Holds the schema of this cursor.

Syntax

Visual Basic
Public Readonly Property **Schema** As ULCursorSchema

C#
public ULCursorSchema **Schema** { get;}

Property value

For result sets, the ULResultSetSchema object representing the schema of the result set. For tables, the ULTableSchema object representing the schema of the table.

Remarks

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

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262
- “ULTableSchema class” on page 542
- “GetSchemaTable method” on page 288
- “ULResultSetSchema class” on page 425

Close method

Closes the cursor.

Syntax

Visual Basic
Public Overrides Sub **Close()**

C#
public override void **Close();**

Remarks

It is not an error to close a cursor that is already closed.

See also

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262

GetBoolean method

Returns the value for the specified column as a System.Boolean.

Syntax

Visual Basic
Public Overrides Function **GetBoolean(** _
  ByVal *columnID* As Integer _
) As Boolean

C#
public override bool **GetBoolean(**
  int *columnID*
);

Parameters

- **columnID** 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.

Return value

The column value as a System.Boolean.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- Boolean
- "FieldCount property" on page 266

GetByte method

Returns the value for the specified column as an unsigned 8-bit value (System.Byte).

Syntax

Visual Basic

```
Public Overrides Function GetByte( _
  ByVal columnID As Integer _
) As Byte
```

C#

```
public override byte GetByte(
  int columnID
);
```

Parameters

- **columnID** 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.

Return value

The column value as a System.Byte.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- Byte
- "FieldCount property" on page 266

GetBytes methods

UL Ext.: Returns the value for the specified column as an array of System.Bytes. Only valid for columns of type ULDbType.Binary, ULDbType.LongBinary, or ULDbType.UniqueIdentifier.

GetBytes(Int32, Int64, Byte[], Int32, Int32) 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.

Syntax

Visual Basic

```
Public Overrides Function GetBytes( _
   ByVal columnID As Integer, _
   ByVal srcOffset As Long, _
   ByVal dst As Byte(), _
   ByVal dstOffset As Integer, _
   ByVal count As Integer _
) As Long
```

C#

```
public override long GetBytes(
   int columnID,
   long srcOffset,
   byte[] dst,
   int dstOffset,
   int count
);
```

Parameters

- **columnID** 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.

Return value

The actual number of bytes copied.

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 *dst*.Length.

For other errors, a ULException with the appropriate error code is thrown.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetBytes methods" on page 273
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- "GetBytes(Int32) method" on page 275
- "ULDbType enumeration" on page 301
- Byte
- "ULException class" on page 306
- "ULSQLCode enumeration" on page 445
- "FieldCount property" on page 266

GetBytes(Int32) method

UL Ext.: Returns the value for the specified column as an array of System.Bytes. Only valid for columns of type ULDbType.Binary, ULDbType.LongBinary, or ULDbType.UniqueIdentifier.

Syntax

Visual Basic

```
Public Function GetBytes( _
  ByVal columnID As Integer _
) As Byte()
```

C#

```
public byte[] GetBytes(
  int columnID
);
```

Parameters

- **columnID** 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.

Return value

The column value as an array of System.Bytes.

See also

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262
- “GetBytes methods” on page 273
- “GetOrdinal method” on page 286
- “GetFieldType method” on page 281
- “GetBytes(Int32, Int64, Byte[], Int32, Int32) method” on page 274
- Byte
- “ULDbType enumeration” on page 301
- “ULDbType enumeration” on page 301
- “ULDbType enumeration” on page 301

GetChar method

This method is not supported in UltraLite.NET.

Syntax

Visual Basic

```
Public Overrides Function GetChar( _
  ByVal columnID As Integer _
) As Char
```

C#

```
public override char GetChar(
  int columnID
);
```

Parameters

- **columnID** 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.

Return value

This method is not supported in UltraLite.NET.

See also

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262
- “GetOrdinal method” on page 286
- “GetFieldType method” on page 281
- “GetString method” on page 290
- “FieldCount property” on page 266

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.

Syntax

Visual Basic

```
Public Overrides Function GetChars( _
  ByVal columnID As Integer, _
  ByVal srcOffset As Long, _
  ByVal dst As Char(), _
  ByVal dstOffset As Integer, _
  ByVal count As Integer _
) As Long
```

C#

```
public override long GetChars(
  int columnID,
  long srcOffset,
  char[] dst,
  int dstOffset,
  int count
);
```

Parameters

- **columnID** 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.

Return value

The actual number of characters copied.

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 *dst*.Length.

For other errors, a ULException with the appropriate error code is thrown.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- "ULDbType enumeration" on page 301
- Char
- "ULException class" on page 306
- "ULSQLCode enumeration" on page 445
- "FieldCount property" on page 266

GetDataTypeName method

Returns the name of the specified column's provider data type.

Syntax

Visual Basic

Public Overrides Function **GetDataTypeName(** _
  ByVal *columnID* As Integer _
) As String

C#

public override string **GetDataTypeName(**
  int *columnID*
);

Parameters

- **columnID** 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.

Return value

A string corresponding to the column's ULDbType.

See also

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262
- “GetOrdinal method” on page 286
- “GetColumnULDbType method” on page 227
- “FieldCount property” on page 266
- “ULDbType enumeration” on page 301

GetDateTime method

Returns the value for the specified column as a System.DateTime with millisecond accuracy.

Syntax

Visual Basic
Public Overrides Function **GetDateTime(** _
  ByVal *columnID* As Integer _
) As Date

C#
public override DateTime **GetDateTime(**
  int *columnID*
);

Parameters

- **columnID** 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.

Return value

The column value as a System.DateTime.

See also

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262
- “GetOrdinal method” on page 286
- “GetFieldType method” on page 281
- DateTime
- “FieldCount property” on page 266

GetDecimal method

Returns the value for the specified column as a System.Decimal.

Syntax

Visual Basic
Public Overrides Function **GetDecimal(** _

ByVal *columnID* As Integer _
) As Decimal

C#
public override decimal **GetDecimal(**
int *columnID*
);

Parameters

- **columnID** 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.

Return value

The column value as a System.Decimal.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- Decimal
- "FieldCount property" on page 266

GetDouble method

Returns the value for the specified column as a System.Double.

Syntax

Visual Basic
Public Overrides Function **GetDouble(** _
ByVal *columnID* As Integer _
) As Double

C#
public override double **GetDouble(**
int *columnID*
);

Parameters

- **columnID** 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.

Return value

The column value as a System.Double.

See also

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262
- “GetOrdinal method” on page 286
- “GetFieldType method” on page 281
- Double
- “FieldCount property” on page 266

GetEnumerator method

Returns an System.Collections.IEnumerator that iterates through the ULDataReader.

Syntax

Visual Basic
Public Overrides Function **GetEnumerator()** As IEnumerator

C#
public override IEnumerator **GetEnumerator();**

Return value

A System.Collections.IEnumerator for the ULDataReader.

See also

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262
- IEnumerator

GetFieldType method

Returns the System.Type most appropriate for the specified column.

Syntax

Visual Basic
Public Overrides Function **GetFieldType(** _
  ByVal *columnID* As Integer _
) As Type

C#
public override Type **GetFieldType(**
  int *columnID*
);

Parameters

- **columnID** 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.

Return value

A System.Type value for the column.

See also

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262
- “GetOrdinal method” on page 286
- “GetDataTypeName method” on page 278
- “GetColumnULDbType method” on page 227
- Type
- “FieldCount property” on page 266

GetFloat method

Returns the value for the specified column as a System.Single.

Syntax

Visual Basic

```
Public Overrides Function GetFloat( _
  ByVal columnID As Integer _
) As Single
```

C#

```
public override float  GetFloat(
  int columnID
);
```

Parameters

- **columnID** 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.

Return value

The column value as a System.Single.

See also

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262
- “GetOrdinal method” on page 286
- “GetFieldType method” on page 281
- Single
- “FieldCount property” on page 266

GetGuid method

Returns the value for the specified column as a UUID (System.Guid).

Syntax

Visual Basic
Public Overrides Function **GetGuid(** _
  ByVal *columnID* As Integer _
) As Guid

C#
public override Guid **GetGuid(**
  int *columnID*
);

Parameters

- **columnID** 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.

Return value

The column value as a Guid.

Remarks

This method is only valid for columns of type ULDbType.UniqueIdentifier or for columns of type ULDbType.Binary with length 16.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- "GetColumnULDbType method" on page 227
- "GetColumnSize method" on page 226
- Guid
- "ULDbType enumeration" on page 301
- "ULDbType enumeration" on page 301
- "FieldCount property" on page 266

GetInt16 method

Returns the value for the specified column as a System.Int16.

Syntax

Visual Basic
Public Overrides Function **GetInt16(** _
  ByVal *columnID* As Integer _
) As Short

C#
public override short **GetInt16(**
  int *columnID*
);

Parameters

- **columnID** 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.

Return value

The column value as an System.Int16.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- Int16
- "FieldCount property" on page 266

GetInt32 method

Returns the value for the specified column as an Int32.

Syntax

Visual Basic

```
Public Overrides Function GetInt32( _
  ByVal columnID As Integer _
) As Integer
```

C#

```
public override int GetInt32(
  int columnID
);
```

Parameters

- **columnID** 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.

Return value

The column value as an Int32.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- Int32
- "FieldCount property" on page 266

GetInt64 method

Returns the value for the specified column as an Int64.

Syntax

Visual Basic

```
Public Overrides Function GetInt64( _
  ByVal columnID As Integer _
) As Long
```

C#

```
public override long GetInt64(
  int columnID
);
```

Parameters

- **columnID** 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.

Return value

The column value as an Int64.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- Int64
- "FieldCount property" on page 266

GetName method

Returns the name of the specified column.

Syntax

Visual Basic

```
Public Overrides Function GetName( _
  ByVal columnID As Integer _
) As String
```

C#

```
public override string  GetName(
  int columnID
);
```

Parameters

- **columnID** 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.

Return value

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.

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

- “ULDataReader class” on page 261
- “ULDataReader members” on page 262
- “FieldCount property” on page 266
- “GetSchemaTable method” on page 288
- “GetColumnName method” on page 223
- “FieldCount property” on page 266

GetOrdinal method

Returns the column ID of the named column.

Syntax

Visual Basic

```
Public Overrides Function GetOrdinal( _
  ByVal columnName As String _
) As Integer
```

C#

```
public override int GetOrdinal(
  string  columnName
);
```

Parameters

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

Return value

The column ID of the named column.

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

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetSchemaTable method" on page 288
- "FieldCount property" on page 266
- "GetColumnID method" on page 222

GetRowCount method

Returns a count of the number of rows in a table or result set. A threshold value is specified to avoid enumerating all rows in cases where it is only necessary to determine that there are more than a specific number of rows available. For example, an application that is displaying information from 10 rows may want to know if there are more than 10 rows available to be able to present an option for the user to display more information.

Syntax

Visual Basic
public int **GetRowCount(** _
ByVal *threshold* as Integer **)** _
as Integer

C#
public int **GetRowCount(** int *threshold* **);**

Return value

If the threshold parameter is zero (no threshold), the return result is the number of rows in the table or result set; otherwise, the number of rows in the table or result set is returned up to a maximum of the threshold value.

Remarks

Enumerating all the rows in a table or result set can be an expensive operation for large tables or result sets. If an application only needs to know if there are more than a specific number of rows, this method can be used.

GetSchemaTable method

Returns a System.Data.DataTable that describes the column metadata of the ULDataReader.

Syntax

Visual Basic
Public Overrides Function **GetSchemaTable()** As DataTable

C#
public override DataTable **GetSchemaTable();**

Return value

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.
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.

DataTable column	Description
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.
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

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "Schema property" on page 271
- "FieldCount property" on page 266
- "ULDbType enumeration" on page 301
- "ULDbType enumeration" on page 301
- "ULDbType enumeration" on page 301
- "ULDbType enumeration" on page 301
- "ULDbType enumeration" on page 301
- DataTable

GetString method

Returns the value for the specified column as a System.String.

Syntax

Visual Basic
Public Overrides Function **GetString(** _
  ByVal *columnID* As Integer _
) As String

C#
public override string **GetString(**
  int *columnID*
);

Parameters

- **columnID** 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.

Return value

The column value as a System.String.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- String
- "FieldCount property" on page 266

GetTimeSpan method

Returns the value for the specified column as a System.TimeSpan with millisecond accuracy.

Syntax

Visual Basic
Public Function **GetTimeSpan(** _
  ByVal *columnID* As Integer _
) As TimeSpan

C#
public TimeSpan **GetTimeSpan(**
  int *columnID*
);

Parameters

- **columnID** 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.

Return value

The column value as a System.TimeSpan.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- TimeSpan
- "FieldCount property" on page 266

GetUInt16 method

Returns the value for the specified column as a System.UInt16.

Syntax

Visual Basic
Public Function **GetUInt16(** _
  ByVal *columnID* As Integer _
) As UInt16

C#
public ushort **GetUInt16(**
  int *columnID*
);

Parameters

- **columnID** 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.

Return value

The column value as an System.UInt16.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- UInt16
- "FieldCount property" on page 266

GetUInt32 method

Returns the value for the specified column as a UInt32.

Syntax

Visual Basic

```
Public Function GetUInt32( _
  ByVal columnID As Integer _
) As UInt32
```

C#

```
public uint GetUInt32(
  int columnID
);
```

Parameters

- **columnID** 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.

Return value

The column value as an UInt32.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- UInt32
- "FieldCount property" on page 266

GetUInt64 method

Returns the value for the specified column as a System.UInt64.

Syntax

Visual Basic

```
Public Function GetUInt64( _
  ByVal columnID As Integer _
) As UInt64
```

C#

```
public ulong GetUInt64(
  int columnID
);
```

Parameters

- **columnID** 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.

Return value

The column value as a System.UInt64

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- UInt64
- "FieldCount property" on page 266

GetValue method

Returns the value of the specified column in its native format.

Syntax

Visual Basic

```
Public Overrides Function GetValue( _
  ByVal columnID As Integer _
) As Object
```

C#

```
public override object GetValue(
  int columnID
);
```

Parameters

- **columnID** 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.

Return value

The column value as the .NET type most appropriate for the column or DBNull if column is NULL.

Remarks

This method is identical in functionality to the ULDataReader.this[int].

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- "FieldCount property" on page 266
- "Item(Int32) property" on page 268

GetValues method

Returns all the column values for the current row.

Syntax

Visual Basic

```
Public Overrides Function GetValues( _
  ByVal values As Object() _
) As Integer
```

C#

```
public override int GetValues(
  object[] values
);
```

Parameters

- **values** The array of System.Objects to hold the entire row.

Return value

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.

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

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "FieldCount property" on page 266
- "GetFieldType method" on page 281
- "GetValue method" on page 293
- Object

IsDBNull method

Checks whether the value from the specified column is NULL.

Syntax

Visual Basic
Public Overrides Function **IsDBNull(** _
  ByVal *columnID* As Integer _
) As Boolean

C#
public override bool **IsDBNull(**
  int *columnID*
);

Parameters

- **columnID** 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.

Return value

True if value is NULL, false if value is not NULL.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- "FieldCount property" on page 266

MoveAfterLast method

UL Ext.: Positions the cursor to after the last row of the cursor.

Syntax

Visual Basic
Public Sub **MoveAfterLast()**

C#
public void **MoveAfterLast();**

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

MoveBeforeFirst method

UL Ext.: Positions the cursor to before the first row of the cursor.

Syntax

Visual Basic
Public Sub **MoveBeforeFirst()**

C#
public void **MoveBeforeFirst();**

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

MoveFirst method

UL Ext.: Positions the cursor to the first row of the cursor.

Syntax

Visual Basic
Public Function **MoveFirst()** As Boolean

C#
public bool **MoveFirst();**

Return value

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

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

MoveLast method

UL Ext.: Positions the cursor to the last row of the cursor.

Syntax

Visual Basic
Public Function **MoveLast()** As Boolean

C#
public bool **MoveLast();**

Return value

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

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

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.

Syntax

Visual Basic
Public Function **MoveNext()** As Boolean

C#
public bool **MoveNext();**

Return value

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

Remarks

This method is identical to the ULDataReader.Read method.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "Read method" on page 299

MovePrevious method

UL Ext.: Positions the cursor to the previous row or before the first row.

Syntax

Visual Basic
Public Function **MovePrevious()** As Boolean

C#
public bool **MovePrevious();**

Return value

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

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

MoveRelative method

UL Ext.: Positions the cursor relative to the current row.

Syntax

Visual Basic
Public Function **MoveRelative(** _
  ByVal *offset* As Integer _
) As Boolean

C#
public bool **MoveRelative(**
  int *offset*
);

Parameters

- **offset** The number of rows to move. Negative values correspond to moving backward.

Return value

True if successful, false otherwise. For example, the method fails if it positions beyond the first or last row.

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

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "IsEOF property" on page 268
- "IsBOF property" on page 267

NextResult method

Advances the ULDataReader to the next result when reading the results of batch SQL statements.

Syntax

Visual Basic
Public Overrides Function **NextResult()** As Boolean

C#
public override bool **NextResult();**

Return value

True if there are more result sets, false otherwise. For UltraLite.NET, always returns false.

Remarks

UL Ext.: UltraLite.NET does not support batches of SQL statements; the ULDataReader is always positioned on the first and only result set. Calling NextResult has no effect.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262

Read method

Positions the cursor to the next row, or after the last row if the cursor was already on the last row.

Syntax

Visual Basic
Public Overrides Function **Read()** As Boolean

C#
public override bool **Read();**

Return value

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

Remarks

This method is identical to the ULDataReader.MoveNext method.

See also

- "ULDataReader class" on page 261
- "ULDataReader members" on page 262
- "MoveNext method" on page 297

ULDateOrder enumeration

UL Ext.: Enumerates the date orders that a database can support.

Syntax

Visual Basic
Public Enum **ULDateOrder**

C#
public enum **ULDateOrder**

Members

Member name	Description	Value
DMY	Day of the month followed by month, followed by year.	2
MDY	Month followed by day of the month, followed by year.	1
YMD	Year followed by month, followed by day of the month.	0

See also

- "DateOrder property" on page 213

ULDbType enumeration

Enumerates the UltraLite.NET database data types.

Syntax

Visual Basic
Public Enum **ULDbType**

C#
public enum **ULDbType**

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
Integer	System.Int32	int	Integer
SmallInt	System.Int16	short	Short
Time	System.TimeSpan	TimeSpan No built-in type.	TimeSpan No built-in type.

ULDbType	Compatible .NET type	C# built-in type	Visual Basic built-in type
DateTime, **TimeStamp**	System.DateTime	DateTime No built-in type.	Date
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.

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

Member name	Description	Value
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
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
Time	Time information.	11
TimeStamp	Timestamp information (date, time). The enumeration values **DateTime** and **TimeStamp** are aliases of each other.	9
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
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

Member name	Description	Value
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

See also

- "GetFieldType method" on page 281
- "GetDataTypeName method" on page 278
- "GetColumnULDbType method" on page 227
- Byte
- Guid
- Boolean
- String
- DateTime
- Single
- Int64
- Int32
- Int16
- TimeSpan
- UInt64
- UInt32
- UInt16

ULDBValid enumeration

Enumerates the UltraLite.NET database validation methods

Syntax

Visual Basic
Public Enum **ULDBValid**

C#
public enum **ULDBValid**

Remarks

For information about validating databases, see "ValidateDatabase method" on page 250.

Members

Member name	Value
EXPRESS_VALIDATE	0
FULL_VALIDATE	1

See also

- "ValidateDatabase method" on page 250
- "ValidateDatabase method" on page 175

ULException class

Represents a SQL error returned by the UltraLite.NET database. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULException**
  Inherits ApplicationException

C#
public sealed class **ULException**: ApplicationException

Remarks

The SQLCODE denoting the error is returned in the NativeError.

This class is not serializable under the .NET Compact Framework.

See also

- "ULException members" on page 306
- "NativeError property" on page 308

ULException members

Public constructors

Member name	Description
"ULException constructor" on page 307	Creates a ULException with the given error code.

Public properties

Member name	Description
Data (inherited from Exception)	Gets a collection of key/value pairs that provide additional user-defined information about the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exception instance that caused the current exception.
Message (inherited from Exception)	Gets a message that describes the current exception.

Member name	Description
"NativeError property" on page 308	Returns the SQL code returned by the database.
"Source property" on page 309	Returns the name of the provider that generated the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public methods

Member name	Description
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
"GetObjectData method" on page 309	Populates a SerializationInfo with the data needed to serialize this ULException.
GetType (inherited from Exception)	Gets the runtime type of the current instance.
ToString (inherited from Exception)	Creates and returns a string representation of the current exception.

See also

- "ULException class" on page 306
- "NativeError property" on page 308

ULException constructor

Creates a ULException with the given error code.

Syntax

Visual Basic

```
Public Sub New( _
  ByVal code As ULSQLCode, _
  ByVal s1 As String, _
  ByVal s2 As String, _
  ByVal s3 As String _
)
```

C#
```
public ULException(
   ULSQLCode code,
   string s1,
   string s2,
   string s3
);
```

Parameters

- **code** The code of the exception.
- **s1** The first string for the formatted message.
- **s2** The second string for the formatted message.
- **s3** The third string for the formatted message.

Remarks

The message string corresponding to the specified iAnywhere.Data.UltraLite.ULSQLCode is retrieved from the **iAnywhere.Data.UltraLite.resources** assembly. Resources are searched for, by culture, using the following order: CultureInfo.CurrentUICulture, then CultureInfo.CurrentCulture, and finally culture "EN".

See also

- "ULException class" on page 306
- "ULException members" on page 306
- "ULSQLCode enumeration" on page 445
- CultureInfo.CurrentUICulture
- CultureInfo.CurrentCulture

NativeError property

Returns the SQLCODE returned by the database.

Syntax

Visual Basic
```
Public Readonly Property NativeError As ULSQLCode
```

C#
```
public ULSQLCode NativeError { get;}
```

Property value

The ULSQLCode value returned by the database.

See also

- "ULException class" on page 306
- "ULException members" on page 306
- "ULSQLCode enumeration" on page 445

Source property

Returns the name of the provider that generated the error.

Syntax

Visual Basic
Public Overrides Readonly Property **Source** As String

C#
public override string **Source** { get;}

Property value

The string value identifying UltraLite.NET as the provider.

See also

- "ULException class" on page 306
- "ULException members" on page 306

GetObjectData method

Populates a SerializationInfo with the data needed to serialize this ULException.

Syntax

Visual Basic
Public Overrides Sub **GetObjectData(** _
  ByVal *info* As SerializationInfo, _
  ByVal *context* As StreamingContext _
)

C#
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.

See also

- "ULException class" on page 306
- "ULException members" on page 306

ULFactory class

Represents a set of methods for creating instances of the iAnywhere.Data.UltraLite provider's implementation of the data source classes. This is a static class and so cannot be inherited or instantiated.

Syntax

Visual Basic
Public NotInheritable Class **ULFactory**
  Inherits DbProviderFactory

C#
public sealed class **ULFactory**: DbProviderFactory

Remarks

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()

// 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().

Restrictions: The ULFactory class is not available in the .NET Compact Framework 2.0.

Inherits: System.Data.Common.DbProviderFactory

See also

- "ULFactory members" on page 311
- DbProviderFactories
- DbProviderFactory

ULFactory members

Public fields

Member name	Description
"Instance field" on page 312	Represents the singleton instance of the ULFactory class. This field is read-only.

Public properties

Member name	Description
"CanCreateDataSourceEnumerator property" on page 312	Returns false to indicate the UltraLite.NET does not support DbDataSourceEnumerator.

Public methods

Member name	Description
"CreateCommand method" on page 313	Returns a strongly typed System.Data.Common.DbCommand instance.
"CreateCommandBuilder method" on page 313	Returns a strongly typed System.Data.Common.DbCommandBuilder instance.
"CreateConnection method" on page 314	Returns a strongly typed System.Data.Common.DbConnection instance.
"CreateConnectionStringBuilder method" on page 314	Returns a strongly typed System.Data.Common.DbConnectionStringBuilder instance.
"CreateDataAdapter method" on page 314	Returns a strongly typed System.Data.Common.DbDataAdapter instance.
CreateDataSourceEnumerator (inherited from DbProviderFactory)	Returns a new instance of the provider's class that implements the DbDataSourceEnumerator.
"CreateParameter method" on page 315	Returns a strongly typed System.Data.Common.DbParameter instance.
CreatePermission (inherited from DbProviderFactory)	Returns a new instance of the provider's class that implements the provider's version of the CodeAccessPermission.

See also

- "ULFactory class" on page 310
- DbProviderFactories
- DbProviderFactory

Instance field

Represents the singleton instance of the ULFactory class. This field is read-only.

Syntax

Visual Basic
Public Shared Readonly **Instance** As ULFactory

C#
public const ULFactory **Instance**;

Remarks

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). For an example, see "ULFactory class" on page 310.

Restrictions: The ULFactory class is not available in the .NET Compact Framework 2.0.

See also

- "ULFactory class" on page 310
- "ULFactory members" on page 311
- DbProviderFactories.GetFactory

CanCreateDataSourceEnumerator property

Returns false to indicate the UltraLite.NET does not support DbDataSourceEnumerator.

Syntax

Visual Basic
Public Overrides Readonly Property **CanCreateDataSourceEnumerator** As Boolean

C#
public override bool **CanCreateDataSourceEnumerator** { get;}

Property value

False to indicate that ULFactory does not implement the CreateDataSourceEnumerator() method.

See also

- “ULFactory class” on page 310
- “ULFactory members” on page 311

CreateCommand method

Returns a strongly typed System.Data.Common.DbCommand instance.

Syntax

Visual Basic
Public Overrides Function **CreateCommand()** As DbCommand

C#
public override DbCommand **CreateCommand();**

Return value

A new ULCommand instance typed as DbCommand.

See also

- “ULFactory class” on page 310
- “ULFactory members” on page 311
- DbCommand
- “ULCommand class” on page 86

CreateCommandBuilder method

Returns a strongly typed System.Data.Common.DbCommandBuilder instance.

Syntax

Visual Basic
Public Overrides Function **CreateCommandBuilder()** As DbCommandBuilder

C#
public override DbCommandBuilder **CreateCommandBuilder();**

Return value

A new ULCommandBuilder instance typed as DbCommandBuilder.

See also

- “ULFactory class” on page 310
- “ULFactory members” on page 311
- DbCommandBuilder
- “ULCommandBuilder class” on page 121

CreateConnection method

Returns a strongly typed System.Data.Common.DbConnection instance.

Syntax

Visual Basic
Public Overrides Function **CreateConnection()** As DbConnection

C#
public override DbConnection **CreateConnection();**

Return value

A new ULConnection instance typed as DbConnection.

See also

- "ULFactory class" on page 310
- "ULFactory members" on page 311
- DbConnection
- "ULConnection class" on page 131

CreateConnectionStringBuilder method

Returns a strongly typed System.Data.Common.DbConnectionStringBuilder instance.

Syntax

Visual Basic
Public Overrides Function **CreateConnectionStringBuilder()** As DbConnectionStringBuilder

C#
public override DbConnectionstring Builder **CreateConnectionStringBuilder();**

Return value

A new ULConnectionStringBuilder instance typed as DbConnectionStringBuilder.

See also

- "ULFactory class" on page 310
- "ULFactory members" on page 311
- DbConnectionStringBuilder
- "ULConnectionStringBuilder class" on page 192

CreateDataAdapter method

Returns a strongly typed System.Data.Common.DbDataAdapter instance.

Syntax

Visual Basic
Public Overrides Function **CreateDataAdapter()** As DbDataAdapter

C#
public override DbDataAdapter **CreateDataAdapter();**

Return value

A new ULDataAdapter instance typed as DbDataAdapter.

See also

- “ULFactory class” on page 310
- “ULFactory members” on page 311
- DbDataAdapter
- “ULDataAdapter class” on page 229

CreateParameter method

Returns a strongly typed System.Data.Common.DbParameter instance.

Syntax

Visual Basic
Public Overrides Function **CreateParameter()** As DbParameter

C#
public override DbParameter **CreateParameter();**

Return value

A new ULParameter instance typed as DbParameter.

See also

- “ULFactory class” on page 310
- “ULFactory members” on page 311
- DbParameter
- “ULParameter class” on page 358

ULFileTransfer class

UL Ext.: Transfers a file from a remote database using the MobiLink server. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULFileTransfer**

C#
public sealed class **ULFileTransfer**

Remarks

You do not need a database connection to perform a file transfer, however, if your application will be using 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

- "ULFileTransfer members" on page 316
- "RuntimeType property" on page 243
- "FileName property" on page 322
- "Stream property" on page 324
- "UserName property" on page 327
- "Version property" on page 327

ULFileTransfer members

Public constructors

Member name	Description
"ULFileTransfer constructor" on page 318	Initializes a ULFileTransfer object. The connection must be opened before you can perform any operations against the database.

Public properties

Member name	Description
"AuthStatus property" on page 319	Returns the authorization status code for the last file transfer attempt.
"AuthValue property" on page 319	Returns the return value from custom user authentication synchronization scripts.

Member name	Description
"AuthenticationParms property" on page 319	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
"DestinationFileName property" on page 320	Specifies the local file name for the downloaded file.
"DestinationPath property" on page 320	Specifies where to download the file.
"DownloadedFile property" on page 321	Checks whether the file was actually downloaded during the last file transfer attempt.
"FileAuthCode property" on page 322	Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.
"FileName property" on page 322	Specifies the name of the file to download.
"ForceDownload property" on page 323	Specifies whether to force the download of the file if it exists.
"Password property" on page 323	The MobiLink password for the user specified by UserName.
"ResumePartialDownload property" on page 324	Specifies whether to resume or discard a previous partial download.
"Stream property" on page 324	Specifies the MobiLink synchronization stream to use for the file transfer.
"StreamErrorCode property" on page 325	Returns the error reported by the stream itself for the last file transfer attempt.
"StreamErrorSystem property" on page 325	Returns the stream error system-specific code.
"StreamParms property" on page 326	Specifies the parameters to configure the synchronization stream.
"UserName property" on page 327	The user name that uniquely identifies the MobiLink client to the MobiLink server.
"Version property" on page 327	Specifies which synchronization script to use.

Public methods

Member name	Description
"DownloadFile methods" on page 328	Download the file specified by the properties of this object.

See also

- "ULFileTransfer class" on page 316
- "RuntimeType property" on page 243
- "FileName property" on page 322
- "Stream property" on page 324
- "UserName property" on page 327
- "Version property" on page 327

ULFileTransfer constructor

Initializes a ULFileTransfer object. The connection must be opened before you can perform any operations against the database.

Syntax

Visual Basic
Public Sub **New()**

C#
public **ULFileTransfer();**

Remarks

You do not need a database connection to perform a file transfer, however, if your application will be using 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

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "DownloadFile() method" on page 328
- "RuntimeType property" on page 243
- "FileName property" on page 322
- "Stream property" on page 324
- "UserName property" on page 327
- "Version property" on page 327

AuthStatus property

Returns the authorization status code for the last file transfer attempt.

Syntax

Visual Basic
Public Readonly Property **AuthStatus** As ULAuthStatusCode

C#
public ULAuthStatusCode **AuthStatus** { get;}

Property value

One of the ULAuthStatusCode values denoting the authorization status for the last file transfer attempt.

See also

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "ULAuthStatusCode enumeration" on page 56

AuthValue property

Returns the return value from custom user authentication synchronization scripts.

Syntax

Visual Basic
Public Readonly Property **AuthValue** As Long

C#
public long **AuthValue** { get;}

Property value

A long integer returned from custom user authentication synchronization scripts.

See also

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Syntax

Visual Basic
Public Property **AuthenticationParms** As String

C#
public string **AuthenticationParms** { get; set; }

Property value

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.

Remarks

Only the first 255 strings are used and each string should be no longer than 128 characters (longer strings are truncated when sent to the MobiLink server).

See also

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316

DestinationFileName property

Specifies the local file name for the downloaded file.

Syntax

Visual Basic
Public Property **DestinationFileName** As String

C#
public string **DestinationFileName** { get; set; }

Property value

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

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "FileName property" on page 322

DestinationPath property

Specifies where to download the file.

Syntax

Visual Basic
Public Property **DestinationPath** As String

C#
public string **DestinationPath** { get; set; }

Property value

A string specifying the destination directory of the file. The default is a null reference (Nothing in Visual Basic).

Remarks

The default destination directory varies depending on the device's operating system:

- For Windows Mobile devices, if the DestinationPath is a null reference (Nothing in Visual Basic), the file is stored in the root (\) directory.
- For desktop applications, if the DestinationPath is a null reference (Nothing in Visual Basic), the file is stored in the current directory.

See also

- “ULFileTransfer class” on page 316
- “ULFileTransfer members” on page 316
- “ForceDownload property” on page 323
- “DownloadFile() method” on page 328

DownloadedFile property

Checks whether the file was actually downloaded during the last file transfer attempt.

Syntax

Visual Basic
Public Readonly Property **DownloadedFile** As Boolean

C#
public bool **DownloadedFile** { get;}

Property value

True if the file was downloaded, false otherwise.

Remarks

If the file is already up-to-date when the DownloadFile() is invoked, it will return true, but DownloadedFile will be false. If an error occurs and DownloadFile() returns false, DownloadedFile will be false.

See also

- “ULFileTransfer class” on page 316
- “ULFileTransfer members” on page 316
- “DownloadFile() method” on page 328
- “DownloadFile() method” on page 328

FileAuthCode property

Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.

Syntax

Visual Basic
Public Readonly Property **FileAuthCode** As UInt16

C#
public ushort **FileAuthCode** { get;}

Property value

An unsigned short integer returned from the authenticate_file_transfer script for the last file transfer attempt.

See also

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316

FileName property

Specifies the name of the file to download.

Syntax

Visual Basic
Public Property **FileName** As String

C#
public string **FileName** { get; set; }

Property value

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.

Remarks

FileName is the name of the file on the server running MobiLink. MobiLink will first search 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, or the MobiLink server will be unable to find it. For example, "myfile.txt" is valid, but "somedir\myfile.txt", "..\myfile.txt", and "c:\myfile.txt" are all invalid.

See also

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "DestinationFileName property" on page 320
- "DownloadFile() method" on page 328

ForceDownload property

Specifies whether to force the download of the file if it exists.

Syntax

Visual Basic
Public Property **ForceDownload** As Boolean

C#
public bool **ForceDownload** { get; set; }

Property value

True to force the download of the file, false to perform normal download checks if file exists. The default is false.

Remarks

If ForceDownload is true, the file will always be downloaded, even if it hasn't changed, and any existing partial download is discarded. If ForceDownload is false, the file will only be downloaded if the server's version of the file is different from the client's. Note that if the file is changed on either the client or the server, specifying ForceDownload true will cause the server version to overwrite the client version.

See also

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "ResumePartialDownload property" on page 324
- "DownloadFile() method" on page 328

Password property

The MobiLink password for the user specified by UserName.

Syntax

Visual Basic
Public Property **Password** As String

C#
public string **Password** { get; set; }

Property value

A string specifying the MobiLink password. The default is a null reference (Nothing in Visual Basic), meaning no password is specified.

Remarks

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

- “ULFileTransfer class” on page 316
- “ULFileTransfer members” on page 316
- “UserName property” on page 327
- “DownloadFile() method” on page 328

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Syntax

Visual Basic
Public Property **ResumePartialDownload** As Boolean

C#
public bool **ResumePartialDownload** { get; set; }

Property value

True to resume a previous partial download, false to discard a previous partial download. The default is false.

Remarks

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 will be discarded and a new download started.

See also

- “ULFileTransfer class” on page 316
- “ULFileTransfer members” on page 316
- “ForceDownload property” on page 323
- “DownloadFile() method” on page 328

Stream property

Specifies the MobiLink synchronization stream to use for the file transfer.

Syntax

Visual Basic
Public Property **Stream** As ULStreamType

C#
public ULStreamType **Stream** { get; set; }

Property value

One of the ULStreamType values specifying the type of synchronization stream to use. The default is ULStreamType.TCPIP.

Remarks

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

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "ULStreamType enumeration" on page 484
- "StreamParms property" on page 326
- "DownloadFile() method" on page 328
- "ULStreamType enumeration" on page 484

StreamErrorCode property

Returns the error reported by the stream itself for the last file transfer attempt.

Syntax

Visual Basic
Public Readonly Property **StreamErrorCode** As ULStreamErrorCode

C#
public ULStreamErrorCode **StreamErrorCode** { get;}

Property value

One of the ULStreamErrorCode values denoting the error reported by the stream itself, ULStreamErrorCode.NONE if no error occurred.

See also

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "ULStreamErrorCode enumeration" on page 462
- "ULStreamErrorCode enumeration" on page 462

StreamErrorSystem property

Returns the stream error system-specific code.

Syntax

Visual Basic
Public Readonly Property **StreamErrorSystem** As Integer

C#
public int **StreamErrorSystem** { get;}

Property value

An integer denoting the stream error system-specific code.

See also

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316

StreamParms property

Specifies the parameters to configure the synchronization stream.

Syntax

Visual Basic
Public Property **StreamParms** As String

C#
public string **StreamParms** { get; set; }

Property value

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).

Remarks

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

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "Stream property" on page 324
- "DownloadFile() method" on page 328
- "ULStreamType enumeration" on page 484

UserName property

The user name that uniquely identifies the MobiLink client to the MobiLink server.

Syntax

Visual Basic
Public Property **UserName** As String

C#
public string **UserName** { get; set; }

Property value

A string specifying the user name. This property has no default value, and must be explicitly set.

Remarks

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

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "Password property" on page 323
- "DownloadFile() method" on page 328

Version property

Specifies which synchronization script to use.

Syntax

Visual Basic
Public Property **Version** As String

C#
public string **Version** { get; set; }

Property value

A string specifying the version of the synchronization script to use. This property has no default value, and must be explicitly set.

Remarks

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

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "DownloadFile() method" on page 328

DownloadFile methods

Download the file specified by the properties of this object.

DownloadFile() method

Download the file specified by the properties of this object.

Syntax

Visual Basic
Public Function **DownloadFile()** As Boolean

C#
public bool **DownloadFile();**

Return value

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.DestinationPath using the ULFileTransfer.Stream, ULFileTransfer.UserName, ULFileTransfer.Password, and ULFileTransfer.Version. Other properties that affect the download are ULFileTransfer.DestinationFileName, ULFileTransfer.AuthenticationParms, ULFileTransfer.ForceDownload and ULFileTransfer.ResumePartialDownload.

To avoid file corruption, UltraLite.NET will download to a temporary file and only replace the destination file once the download has completed.

A detailed result status is reported in this object's ULFileTransfer.AuthStatus, ULFileTransfer.AuthValue, ULFileTransfer.FileAuthCode, ULFileTransfer.DownloadedFile, ULFileTransfer.StreamErrorCode, and ULFileTransfer.StreamErrorSystem.

See also

- "ULFileTransfer class" on page 316
- "ULFileTransfer members" on page 316
- "DownloadFile methods" on page 328
- "DownloadFile(ULFileTransferProgressListener) method" on page 329
- "FileName property" on page 322
- "DestinationPath property" on page 320
- "Stream property" on page 324
- "UserName property" on page 327
- "Password property" on page 323
- "Version property" on page 327
- "DestinationFileName property" on page 320
- "AuthenticationParms property" on page 319
- "ForceDownload property" on page 323
- "ResumePartialDownload property" on page 324
- "AuthStatus property" on page 319
- "AuthValue property" on page 319
- "FileAuthCode property" on page 322
- "DownloadedFile property" on page 321
- "StreamErrorCode property" on page 325
- "StreamErrorSystem property" on page 325
- "StreamErrorCode property" on page 325

DownloadFile(ULFileTransferProgressListener) method

Download the file specified by the properties of this object with progress events posted to the specified listener.

Syntax

Visual Basic

```
Public Function DownloadFile( _
  ByVal listener As ULFileTransferProgressListener _
) As Boolean
```

C#

```
public bool DownloadFile(
  ULFileTransferProgressListener listener
);
```

Parameters

- **listener** The object that receives file transfer progress events.

Return value

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.DestinationPath using the ULFileTransfer.Stream, ULFileTransfer.UserName, ULFileTransfer.Password, and ULFileTransfer.Version. Other properties that affect the download are ULFileTransfer.DestinationFileName, ULFileTransfer.AuthenticationParms, ULFileTransfer.ForceDownload and ULFileTransfer.ResumePartialDownload.

To avoid file corruption, UltraLite.NET will download to a temporary file and only replace the destination file once the download has completed.

A detailed result status is reported in this object's ULFileTransfer.AuthStatus, ULFileTransfer.AuthValue, ULFileTransfer.FileAuthCode, ULFileTransfer.DownloadedFile, ULFileTransfer.StreamErrorCode, and ULFileTransfer.StreamErrorSystem.

Errors may result in no data being sent to the listener.

See also

- “ULFileTransfer class” on page 316
- “ULFileTransfer members” on page 316
- “DownloadFile methods” on page 328
- “FileName property” on page 322
- “DestinationPath property” on page 320
- “Stream property” on page 324
- “UserName property” on page 327
- “Password property” on page 323
- “Version property” on page 327
- “DestinationFileName property” on page 320
- “AuthenticationParms property” on page 319
- “ForceDownload property” on page 323
- “ResumePartialDownload property” on page 324
- “AuthStatus property” on page 319
- “AuthValue property” on page 319
- “FileAuthCode property” on page 322
- “DownloadedFile property” on page 321
- “StreamErrorCode property” on page 325
- “StreamErrorSystem property” on page 325
- “StreamErrorCode property” on page 325

ULFileTransferProgressData class

UL Ext.: Returns file transfer progress monitoring data.

Syntax

Visual Basic
Public Class **ULFileTransferProgressData**

C#
public class **ULFileTransferProgressData**

See also

- "ULFileTransferProgressData members" on page 331
- "ULFileTransferProgressListener interface" on page 334

ULFileTransferProgressData members

Public fields

Member name	Description
"FLAG_IS_BLOCKING field" on page 332	A flag indicating that the file transfer is blocked awaiting a response from the MobiLink server. This field is constant and read-only.

Public properties

Member name	Description
"BytesReceived property" on page 332	Returns the number of bytes received so far.
"FileSize property" on page 332	Returns the size of the file being transferred.
"Flags property" on page 333	Returns the current file transfer flags indicating additional information relating to the current state.
"ResumedAtSize property" on page 333	Returns the point in the file where the transfer was resumed.

See also

- "ULFileTransferProgressData class" on page 331
- "ULFileTransferProgressListener interface" on page 334

FLAG_IS_BLOCKING field

A flag indicating that the file transfer is blocked awaiting a response from the MobiLink server. This field is constant and read-only.

Syntax

Visual Basic
Public Shared **FLAG_IS_BLOCKING** As Integer

C#
public const int **FLAG_IS_BLOCKING**;

See also

- "ULFileTransferProgressData class" on page 331
- "ULFileTransferProgressData members" on page 331

BytesReceived property

Returns the number of bytes received so far.

Syntax

Visual Basic
Public Readonly Property **BytesReceived** As UInt64

C#
public ulong **BytesReceived** { get;}

Property value

The number of bytes received so far.

See also

- "ULFileTransferProgressData class" on page 331
- "ULFileTransferProgressData members" on page 331

FileSize property

Returns the size of the file being transferred.

Syntax

Visual Basic
Public Readonly Property **FileSize** As UInt64

C#
public ulong **FileSize** { get;}

Property value

The size of the file in bytes.

See also

- “ULFileTransferProgressData class” on page 331
- “ULFileTransferProgressData members” on page 331

Flags property

Returns the current file transfer flags indicating additional information relating to the current state.

Syntax

Visual Basic
Public Readonly Property **Flags** As Integer

C#
public int **Flags** { get;}

Property value

An integer containing a combination of flags or'ed together.

See also

- “ULFileTransferProgressData class” on page 331
- “ULFileTransferProgressData members” on page 331
- “FLAG_IS_BLOCKING field” on page 332

ResumedAtSize property

Returns the point in the file where the transfer was resumed.

Syntax

Visual Basic
Public Readonly Property **ResumedAtSize** As UInt64

C#
public ulong **ResumedAtSize** { get;}

Property value

The number of bytes transferred previously.

See also

- “ULFileTransferProgressData class” on page 331
- “ULFileTransferProgressData members” on page 331

ULFileTransferProgressListener interface

UL Ext.: The listener interface for receiving file transfer progress events.

Syntax

Visual Basic

Public Interface **ULFileTransferProgressListener**

C#

public interface **ULFileTransferProgressListener**

See also

- "ULFileTransferProgressListener members" on page 334
- "DownloadFile(ULFileTransferProgressListener) method" on page 329

ULFileTransferProgressListener members

Public methods

Member name	Description
"FileTransferProgressed method" on page 334	Invoked during a file transfer to inform the user of progress. This method should return true to cancel the transfer or return false to continue.

See also

- "ULFileTransferProgressListener interface" on page 334
- "DownloadFile(ULFileTransferProgressListener) method" on page 329

FileTransferProgressed method

Invoked during a file transfer to inform the user of progress. This method should return true to cancel the transfer or return false to continue.

Syntax

Visual Basic

```
Public Function FileTransferProgressed( _
  ByVal data As ULFileTransferProgressData _
) As Boolean
```

C#

```
public bool FileTransferProgressed(
  ULFileTransferProgressData data
);
```

Parameters

- **data** A ULFileTransferProgressData object containing the latest file transfer progress data.

Return value

This method should return true to cancel the transfer or return false to continue.

Remarks

No UltraLite.NET API methods should be invoked during a FileTransferProgressed call.

See also

- “ULFileTransferProgressListener interface” on page 334
- “ULFileTransferProgressListener members” on page 334
- “ULFileTransferProgressData class” on page 331

ULIndexSchema class

UL Ext.: Represents the schema of an UltraLite table index. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULIndexSchema**

C#
public sealed class **ULIndexSchema**

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

- "ULIndexSchema members" on page 336
- "PrimaryKey property" on page 546
- "GetIndex method" on page 548
- "GetOptimalIndex method" on page 549
- "ULTableSchema class" on page 542

ULIndexSchema members

Public properties

Member name	Description
"ColumnCount property" on page 337	Returns the number of columns in the index.
"IsForeignKey property" on page 338	Checks whether the index is a foreign key.
"IsForeignKeyCheckOnCommit property" on page 338	Checks whether referential integrity for the foreign key is performed on commits or on inserts and updates.
"IsForeignKeyNullable property" on page 339	Checks whether the foreign key is nullable.
"IsOpen property" on page 339	Determines whether the index schema is open or closed.
"IsPrimaryKey property" on page 340	Checks whether the index is the primary key.

Member name	Description
"IsUniqueIndex property" on page 340	Checks whether the index is unique.
"IsUniqueKey property" on page 341	Checks whether the index is a unique key.
"Name property" on page 341	Returns the name of the index.
"ReferencedIndexName property" on page 341	The name of the referenced primary index if the index is a foreign key.
"ReferencedTableName property" on page 342	The name of the referenced primary table if the index is a foreign key.

Public methods

Member name	Description
"GetColumnName method" on page 342	Returns the name of the *colOrdinalInIndex*'th column in this index.
"IsColumnDescending method" on page 343	Checks whether the named column is used in descending order by the index.

See also

- "ULIndexSchema class" on page 336
- "PrimaryKey property" on page 546
- "GetIndex method" on page 548
- "GetOptimalIndex method" on page 549
- "ULTableSchema class" on page 542

ColumnCount property

Returns the number of columns in the index.

Syntax

Visual Basic
Public Readonly Property **ColumnCount** As Short

C#
public short **ColumnCount** { get;}

Property value

The number of columns in the index.

Remarks

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.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336

IsForeignKey property

Checks whether the index is a foreign key.

Syntax

Visual Basic
Public Readonly Property **IsForeignKey** As Boolean

C#
public bool **IsForeignKey** { get;}

Property value

True if the index is the foreign key, false if the index is not the foreign key.

Remarks

Columns in a foreign key may reference another table's non-null, unique index.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336

IsForeignKeyCheckOnCommit property

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

Syntax

Visual Basic
Public Readonly Property **IsForeignKeyCheckOnCommit** As Boolean

C#
public bool **IsForeignKeyCheckOnCommit** { get;}

Property value

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

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336
- "IsForeignKey property" on page 338

IsForeignKeyNullable property

Checks whether the foreign key is nullable.

Syntax

Visual Basic
Public Readonly Property **IsForeignKeyNullable** As Boolean

C#
public bool **IsForeignKeyNullable** { get;}

Property value

True if the foreign key is nullable, false if the foreign key is not nullable.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336
- "IsForeignKey property" on page 338

IsOpen property

Determines whether the index schema is open or closed.

Syntax

Visual Basic
Public Readonly Property **IsOpen** As Boolean

C#
public bool **IsOpen** { get;}

Property value

True if the index schema is open, otherwise false.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336

IsPrimaryKey property

Checks whether the index is the primary key.

Syntax

Visual Basic
Public Readonly Property **IsPrimaryKey** As Boolean

C#
public bool **IsPrimaryKey** { get;}

Property value

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

Remarks

Columns in the primary key may not be null.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336

IsUniqueIndex property

Checks whether the index is unique.

Syntax

Visual Basic
Public Readonly Property **IsUniqueIndex** As Boolean

C#
public bool **IsUniqueIndex** { get;}

Property value

True if the index is unique, false if the index is not unique.

Remarks

Columns in a unique index may be null.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336

IsUniqueKey property

Checks whether the index is a unique key.

Syntax

Visual Basic
Public Readonly Property **IsUniqueKey** As Boolean

C#
public bool **IsUniqueKey** { get;}

Property value

True if the index is a unique key, false if the index is not a unique key.

Remarks

Columns in a unique key may not be null.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336

Name property

Returns the name of the index.

Syntax

Visual Basic
Public Readonly Property **Name** As String

C#
public string **Name** { get;}

Property value

A string specifying the name of the index.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336

ReferencedIndexName property

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

Syntax

Visual Basic
Public Readonly Property **ReferencedIndexName** As String

C#
public string **ReferencedIndexName** { get;}

Property value

A string specifying the name of the referenced primary index.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336
- "IsForeignKey property" on page 338

ReferencedTableName property

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

Syntax

Visual Basic
Public Readonly Property **ReferencedTableName** As String

C#
public string **ReferencedTableName** { get;}

Property value

A string specifying the name of the referenced primary table.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336
- "IsForeignKey property" on page 338

GetColumnName method

Returns the name of the

colOrdinalInIndex'th column in this index.

Syntax

Visual Basic
Public Function **GetColumnName(** _
  ByVal *colOrdinalInIndex* As Short _
) As String

C#
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].

Return value

The name of the column.

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

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336
- "ColumnCount property" on page 337
- "ColumnCount property" on page 337

IsColumnDescending method

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

Syntax

Visual Basic
Public Function **IsColumnDescending(** _
  ByVal *name* As String _
) As Boolean

C#
public bool **IsColumnDescending(**
  string *name*
);

Parameters

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

Return value

True if the column is used in descending order, false if the column is used in ascending order.

See also

- "ULIndexSchema class" on page 336
- "ULIndexSchema members" on page 336
- "GetColumnName method" on page 342
- "ColumnCount property" on page 337

ULInfoMessageEventArgs class

Provides data for the ULConnection.InfoMessage event. This class cannot be inherited.

Syntax

Visual Basic

```
Public NotInheritable Class ULInfoMessageEventArgs
  Inherits EventArgs
```

C#

```
public sealed class ULInfoMessageEventArgs: EventArgs
```

See also

- "ULInfoMessageEventArgs members" on page 345
- "InfoMessage event" on page 176

ULInfoMessageEventArgs members

Public properties

Member name	Description
"Message property" on page 345	The informational or warning message string returned by the database.
"NativeError property" on page 346	The SQL code corresponding to the informational message or warning returned by the database.
"Source property" on page 346	The name of the ADO.NET data provider returning the message.

Public methods

Member name	Description
"ToString method" on page 347	Returns the string representation of the ULConnection.InfoMessage event.

See also

- "ULInfoMessageEventArgs class" on page 345
- "InfoMessage event" on page 176

Message property

The informational or warning message string returned by the database.

Syntax

Visual Basic
Public Readonly Property **Message** As String

C#
public string **Message** { get;}

Property value

A string containing the informational or warning message.

See also

- "ULInfoMessageEventArgs class" on page 345
- "ULInfoMessageEventArgs members" on page 345

NativeError property

The SQLCODE corresponding to the informational message or warning returned by the database.

Syntax

Visual Basic
Public Readonly Property **NativeError** As ULSQLCode

C#
public ULSQLCode **NativeError** { get;}

Property value

An informational or warning ULSQLCode value.

See also

- "ULInfoMessageEventArgs class" on page 345
- "ULInfoMessageEventArgs members" on page 345
- "ULSQLCode enumeration" on page 445

Source property

The name of the ADO.NET data provider returning the message.

Syntax

Visual Basic
Public Readonly Property **Source** As String

C#
public string **Source** { get;}

Property value

The string "UltraLite.NET Data Provider".

See also

- "ULInfoMessageEventArgs class" on page 345
- "ULInfoMessageEventArgs members" on page 345

ToString method

Returns the string representation of the ULConnection.InfoMessage event.

Syntax

Visual Basic
Public Overrides Function **ToString()** As String

C#
public override string **ToString();**

Return value

The informational or warning message string.

See also

- "ULInfoMessageEventArgs class" on page 345
- "ULInfoMessageEventArgs members" on page 345
- "InfoMessage event" on page 176

ULInfoMessageEventHandler delegate

Represents the method that will handle the ULConnection.InfoMessage event.

Syntax

Visual Basic

```
Public Delegate Sub ULInfoMessageEventHandler( _
  ByVal obj As Object, _
  ByVal args As ULInfoMessageEventArgs _
)
```

C#

```
public delegate void ULInfoMessageEventHandler(
  object obj,
  ULInfoMessageEventArgs args
);
```

See also

- "InfoMessage event" on page 176
- "ULInfoMessageEventArgs class" on page 345

ULMetaDataCollectionNames class

Provides a list of constants for use with the ULConnection.GetSchema(String,String[]) to retrieve metadata collections. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULMetaDataCollectionNames**

C#
public sealed class **ULMetaDataCollectionNames**

See also

- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String, String[]) method" on page 165

ULMetaDataCollectionNames members

Public properties

Member name	Description
"Columns property" on page 350	Provides a constant for use with the ULConnection.GetSchema(String) that represents the Columns collection.
"DataSourceInformation property" on page 351	Provides a constant for use with the ULConnection.GetSchema(String) that represents the DataSourceInformation collection.
"DataTypes property" on page 351	Provides a constant for use with the ULConnection.GetSchema(String) that represents the DataTypes collection.
"ForeignKeys property" on page 352	Provides a constant for use with the ULConnection.GetSchema(String) that represents the ForeignKeys collection.
"IndexColumns property" on page 353	Provides a constant for use with the ULConnection.GetSchema(String) that represents the IndexColumns collection.
"Indexes property" on page 353	Provides a constant for use with the ULConnection.GetSchema(String) that represents the Indexes collection.
"MetaDataCollections property" on page 354	Provides a constant for use with the ULConnection.GetSchema(String) that represents the MetaDataCollections collection.
"Publications property" on page 355	Provides a list of constants for use with the ULConnection.GetSchema(String,String[]) to retrieve metadata collections.

Member name	Description
"ReservedWords property" on page 355	Provides a constant for use with the ULConnection.GetSchema(String) that represents the ReservedWords collection.
"Restrictions property" on page 356	Provides a constant for use with the ULConnection.GetSchema(String) that represents the Restrictions collection.
"Tables property" on page 356	Provides a constant for use with the ULConnection.GetSchema(String) that represents the Tables collection.

See also

- "ULMetaDataCollectionNames class" on page 349
- "GetSchema(String, String[]) method" on page 165

Columns property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the Columns collection.

Syntax

Visual Basic
Public Shared Readonly Property **Columns** As String

C#
public const string **Columns** { get;}

Property value

A string representing the name of the Columns collection.

Example

The following code fills a DataTable with the Columns collection.

```
' Visual Basic
Dim schema As DataTable = _
  conn.GetSchema( ULMetaDataCollectionNames.Columns )

// C#
DataTable schema =
  conn.GetSchema( ULMetaDataCollectionNames.Columns );
```

See also

- "ULMetaDataCollectionNames class" on page 349
- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String) method" on page 164

DataSourceInformation property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the DataSourceInformation collection.

Syntax

Visual Basic
Public Shared Readonly Property **DataSourceInformation** As String

C#
public const string **DataSourceInformation** { get;}

Property value

A string representing the name of the DataSourceInformation collection.

Example

The following code fills a DataTable with the DataSourceInformation collection.

```
' Visual Basic
Dim schema As DataTable = _
  conn.GetSchema( ULMetaDataCollectionNames.DataSourceInformation )

// C#
DataTable schema =
  conn.GetSchema( ULMetaDataCollectionNames.DataSourceInformation );
```

See also

- "ULMetaDataCollectionNames class" on page 349
- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String) method" on page 164

DataTypes property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the DataTypes collection.

Syntax

Visual Basic
Public Shared Readonly Property **DataTypes** As String

C#
public const string **DataTypes** { get;}

Property value

A string representing the name of the DataTypes collection.

Example

The following code fills a DataTable with the DataTypes collection.

```
' Visual Basic
Dim schema As DataTable = _
  conn.GetSchema( ULMetaDataCollectionNames.DataTypes )

// C#
DataTable schema =
  conn.GetSchema( ULMetaDataCollectionNames.DataTypes );
```

See also

- "ULMetaDataCollectionNames class" on page 349
- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String) method" on page 164

ForeignKeys property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the ForeignKeys collection.

Syntax

Visual Basic
Public Shared Readonly Property **ForeignKeys** As String

C#
public const string **ForeignKeys** { get;}

Property value

A string representing the name of the ForeignKeys collection.

Example

The following code fills a DataTable with the ForeignKeys collection.

```
' Visual Basic
Dim schema As DataTable = _
  conn.GetSchema( ULMetaDataCollectionNames.ForeignKeys )

// C#
DataTable schema =
  conn.GetSchema( ULMetaDataCollectionNames.ForeignKeys );
```

See also

- "ULMetaDataCollectionNames class" on page 349
- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String) method" on page 164

IndexColumns property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the IndexColumns collection.

Syntax

Visual Basic
Public Shared Readonly Property **IndexColumns** As String

C#
public const string **IndexColumns** { get;}

Property value

A string representing the name of the IndexColumns collection.

Example

The following code fills a DataTable with the IndexColumns collection.

```
' Visual Basic
Dim schema As DataTable = _
  conn.GetSchema( ULMetaDataCollectionNames.IndexColumns )

// C#
DataTable schema =
  conn.GetSchema( ULMetaDataCollectionNames.IndexColumns );
```

See also

- “ULMetaDataCollectionNames class” on page 349
- “ULMetaDataCollectionNames members” on page 349
- “GetSchema(String) method” on page 164

Indexes property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the Indexes collection.

Syntax

Visual Basic
Public Shared Readonly Property **Indexes** As String

C#
public const string **Indexes** { get;}

Property value

A string representing the name of the Indexes collection.

Example

The following code fills a DataTable with the Indexes collection.

```
' Visual Basic
Dim schema As DataTable = _
  conn.GetSchema( ULMetaDataCollectionNames.Indexes )

// C#
DataTable schema =
  conn.GetSchema( ULMetaDataCollectionNames.Indexes );
```

See also

- "ULMetaDataCollectionNames class" on page 349
- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String) method" on page 164

MetaDataCollections property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the MetaDataCollections collection.

Syntax

Visual Basic
Public Shared Readonly Property **MetaDataCollections** As String

C#
public const string **MetaDataCollections** { get;}

Property value

A string representing the name of the MetaDataCollections collection.

Example

The following code fills a DataTable with the MetaDataCollections collection.

```
' Visual Basic
Dim schema As DataTable = _
  conn.GetSchema( ULMetaDataCollectionNames.MetaDataCollections )

// C#
DataTable schema =
  conn.GetSchema( ULMetaDataCollectionNames.MetaDataCollections );
```

See also

- "ULMetaDataCollectionNames class" on page 349
- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String) method" on page 164

Publications property

Provides a list of constants for use with the ULConnection.GetSchema(String,String[]) to retrieve metadata collections.

Syntax

Visual Basic
Public Shared Readonly Property **Publications** As String

C#
public const string **Publications** { get;}

See also

- "ULMetaDataCollectionNames class" on page 349
- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String, String[]) method" on page 165

ReservedWords property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the ReservedWords collection.

Syntax

Visual Basic
Public Shared Readonly Property **ReservedWords** As String

C#
public const string **ReservedWords** { get;}

Property value

A string representing the name of the ReservedWords collection.

Example

The following code fills a DataTable with the ReservedWords collection.

```
' Visual Basic
Dim schema As DataTable = _
  conn.GetSchema( ULMetaDataCollectionNames.ReservedWords )

// C#
DataTable schema =
  conn.GetSchema( ULMetaDataCollectionNames.ReservedWords );
```

See also

- "ULMetaDataCollectionNames class" on page 349
- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String) method" on page 164

Restrictions property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the Restrictions collection.

Syntax

Visual Basic
Public Shared Readonly Property **Restrictions** As String

C#
public const string **Restrictions** { get;}

Property value

A string representing the name of the Restrictions collection.

Example

The following code fills a DataTable with the Restrictions collection.

```
' Visual Basic
Dim schema As DataTable = _
  conn.GetSchema( ULMetaDataCollectionNames.Restrictions )

// C#
DataTable schema =
  conn.GetSchema( ULMetaDataCollectionNames.Restrictions );
```

See also

- "ULMetaDataCollectionNames class" on page 349
- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String) method" on page 164

Tables property

Provides a constant for use with the ULConnection.GetSchema(String) that represents the Tables collection.

Syntax

Visual Basic
Public Shared Readonly Property **Tables** As String

C#
public const string **Tables** { get;}

Property value

A string representing the name of the Tables collection.

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 );
```

See also

- "ULMetaDataCollectionNames class" on page 349
- "ULMetaDataCollectionNames members" on page 349
- "GetSchema(String) method" on page 164

ULParameter class

Represents a parameter to a ULCommand. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULParameter**
  Inherits DbParameter
  Implements ICloneable

C#
public sealed class **ULParameter**: DbParameter,
  ICloneable

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 ) )

// 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.

Inherits: System.Data.Common.DbParameter

Implements: System.Data.IDbDataParameter, System.Data.IDataParameter

See also

- "ULParameter members" on page 359
- "ULCommand class" on page 86
- "CreateParameter method" on page 104
- "ULParameter(String, Object) constructor" on page 361
- "Parameters property" on page 97
- "Value property" on page 372
- DbParameter
- IDbDataParameter
- IDataParameter

ULParameter members

Public constructors

Member name	Description
"ULParameter constructors" on page 360	Initializes a new instance of the "ULParameter class" on page 358.

Public properties

Member name	Description
"DbType property" on page 366	Specifies the System.Data.DbType of the parameter
"Direction property" on page 366	A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.
"IsNullable property" on page 367	Specifies whether the parameter accepts null values.
"Offset property" on page 368	Specifies the offset to the ULParameter.Value.
"ParameterName property" on page 368	Specifies the name of the parameter.
"Precision property" on page 369	Specifies the maximum number of digits used to represent the ULParameter.Value.
"Scale property" on page 369	Specifies the number of decimal places to which ULParameter.Value is resolved.
"Size property" on page 370	Specifies the maximum size of the data within the column.
"SourceColumn property" on page 370	Specifies the name of the source column mapped to the DataSet and used for loading or returning the value.
"SourceColumnNullMapping property" on page 371	
"SourceVersion property" on page 371	The System.Data.DataRowVersion to use when loading ULParameter.Value.
"ULDbType property" on page 372	Specifies the iAnywhere.Data.UltraLite.ULDbType of the parameter
"Value property" on page 372	Specifies the value of the parameter.

Public methods

Member name	Description
"ResetDbType method" on page 373	This method is not supported in UltraLite.NET.
"ToString method" on page 373	Returns the string representation of this instance.

See also

- "ULParameter class" on page 358
- "ULCommand class" on page 86
- "CreateParameter method" on page 104
- "ULParameter(String, Object) constructor" on page 361
- "Parameters property" on page 97
- "Value property" on page 372
- DbParameter
- IDbDataParameter
- IDataParameter

ULParameter constructors

Initializes a new instance of the "ULParameter class" on page 358.

ULParameter() constructor

Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.

Syntax

Visual Basic
Public Sub **New()**

C#
public **ULParameter();**

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 )

// C#
ULParameter p = new ULParameter();
p.Value = 3;
cmd.Parameters.Add( p );
```

See also

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "ULParameter constructors" on page 360
- "Value property" on page 372
- "ULParameter(String, Object) constructor" on page 361
- "ULCommand class" on page 86

ULParameter(String, Object) constructor

Initializes a ULParameter object with the specified parameter name and value.

Syntax

Visual Basic

Public Sub **New(** _
  ByVal *parameterName* As String, _
  ByVal *value* As Object _
)

C#

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.

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 ) ) )

// C#
cmd.Parameters.Add( new ULParameter( "", (object)0 ) );
```

See also

- “ULParameter class” on page 358
- “ULParameter members” on page 359
- “ULParameter constructors” on page 360
- “ULParameter() constructor” on page 360
- “ULCommand class” on page 86
- Object

ULParameter(String, ULDbType) constructor

Initializes a ULParameter object with the specified parameter name and data type. This constructor is not recommended; it is provided for compatibility with other data providers.

Syntax

Visual Basic

```
Public Sub New( _
  ByVal parameterName As String, _
  ByVal dbType As ULDbType _
)
```

C#

```
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

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 class” on page 358
- “ULParameter members” on page 359
- “ULParameter constructors” on page 360
- “ULParameter() constructor” on page 360
- “ULParameter(String, Object) constructor” on page 361
- “Value property” on page 372
- “ULCommand class” on page 86
- “ULDbType enumeration” on page 301

ULParameter(String, ULDbType, Int32) constructor

Initializes a ULParameter object with the specified parameter name and data type. This constructor is not recommended; it is provided for compatibility with other data providers.

Syntax

Visual Basic
```
Public Sub New( _
  ByVal parameterName As String, _
  ByVal dbType As ULDbType, _
  ByVal size As Integer _
)
```

C#
```
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

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 class” on page 358
- “ULParameter members” on page 359
- “ULParameter constructors” on page 360
- “ULParameter() constructor” on page 360
- “ULParameter(String, Object) constructor” on page 361
- “Value property” on page 372
- “ULDbType enumeration” on page 301

ULParameter(String, ULDbType, Int32, String) constructor

Initializes a ULParameter object with the specified parameter name, data type, and length. This constructor is not recommended; it is provided for compatibility with other data providers.

Syntax

Visual Basic

```
Public Sub New( _
  ByVal parameterName As String, _
  ByVal dbType As ULDbType, _
  ByVal size As Integer, _
  ByVal sourceColumn As String _
)
```

C#

```
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

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 class" on page 358
- "ULParameter members" on page 359
- "ULParameter constructors" on page 360
- "ULParameter() constructor" on page 360
- "ULParameter(String, Object) constructor" on page 361
- "Value property" on page 372
- "ULDbType enumeration" on page 301
- "ULCommand class" on page 86

ULParameter(String, ULDbType, Int32, ParameterDirection, Boolean, 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. This constructor is not recommended; it is provided for compatibility with other data providers.

Syntax

Visual Basic

```
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#

```
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.

See also

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "ULParameter constructors" on page 360
- "ULParameter() constructor" on page 360
- "ULParameter(String, Object) constructor" on page 361
- "ULDbType enumeration" on page 301
- ParameterDirection
- DataRowVersion
- Object
- ParameterDirection.Input
- "ULCommand class" on page 86

DbType property

Specifies the System.Data.DbType of the parameter

Syntax

Visual Basic
Public Overrides Property **DbType** As DbType

C#
public override DbType **DbType** { get; set; }

Property value

One of the System.Data.DbType values.

Remarks

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

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- DbType
- DbType
- "ULDbType property" on page 372
- "ULDbType enumeration" on page 301

Direction property

A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

Syntax

Visual Basic
Public Overrides Property **Direction** As ParameterDirection

C#
public override ParameterDirection **Direction** { get; set; }

Property value

One of the System.Data.ParameterDirection values.

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

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- ParameterDirection
- ParameterDirection.Input
- "Value property" on page 372

IsNullable property

Specifies whether the parameter accepts null values.

Syntax

Visual Basic
Public Overrides Property **IsNullable** As Boolean

C#
public override bool **IsNullable** { get; set; }

Property value

True if null values are accepted, false otherwise. The default is false. Null values are handled using the DBNull class.

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

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "Value property" on page 372

Offset property

Specifies the offset to the ULParameter.Value.

Syntax

Visual Basic
Public Property **Offset** As Integer

C#
public int **Offset** { get; set; }

Property value

The offset to the value. The default is 0.

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

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "Value property" on page 372

ParameterName property

Specifies the name of the parameter.

Syntax

Visual Basic
Public Overrides Property **ParameterName** As String

C#
public override string **ParameterName** { get; set; }

Property value

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.

Remarks

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

- “ULParameter class” on page 358
- “ULParameter members” on page 359
- “ULCommand class” on page 86

Precision property

Specifies the maximum number of digits used to represent the ULParameter.Value.

Syntax

Visual Basic
Public Property **Precision** As Byte

C#
public byte **Precision** { get; set; }

Property value

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.

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

- “ULParameter class” on page 358
- “ULParameter members” on page 359
- “Value property” on page 372

Scale property

Specifies the number of decimal places to which ULParameter.Value is resolved.

Syntax

Visual Basic
Public Property **Scale** As Byte

C#
public byte **Scale** { get; set; }

Property value

The number of decimal places to which ULParameter.Value is resolved. The default is 0.

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

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "Value property" on page 372

Size property

Specifies the maximum size of the data within the column.

Syntax

Visual Basic
Public Overrides Property **Size** As Integer

C#
public override int **Size** { get; set; }

Property value

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.

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

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "Value property" on page 372

SourceColumn property

Specifies the name of the source column mapped to the DataSet and used for loading or returning the value.

Syntax

Visual Basic
Public Overrides Property **SourceColumn** As String

C#
public override string **SourceColumn** { get; set; }

Property value

A string specifying the name of the source column mapped to the DataSet and used for loading or returning the value.

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

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "Value property" on page 372

SourceColumnNullMapping property

Syntax

Visual Basic
Public Overrides Property **SourceColumnNullMapping** As Boolean

C#
public override bool **SourceColumnNullMapping** { get; set; }

See also

- "ULParameter class" on page 358
- "ULParameter members" on page 359

SourceVersion property

The System.Data.DataRowVersion to use when loading ULParameter.Value.

Syntax

Visual Basic
Public Overrides Property **SourceVersion** As DataRowVersion

C#
public override DataRowVersion **SourceVersion** { get; set; }

See also

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "Value property" on page 372
- DataRowVersion

ULDbType property

Specifies the iAnywhere.Data.UltraLite.ULDbType of the parameter

Syntax

Visual Basic
Public Property **ULDbType** As ULDbType

C#
public ULDbType **ULDbType** { get; set; }

Property value

One of the iAnywhere.Data.UltraLite.ULDbType values.

Remarks

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

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "Value property" on page 372
- "DbType property" on page 366
- DbType
- "ULDbType enumeration" on page 301

Value property

Specifies the value of the parameter.

Syntax

Visual Basic
Public Overrides Property **Value** As Object

C#
public override object **Value** { get; set; }

Property value

A System.Object that specifies the value of the parameter.

Remarks

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

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "ULException class" on page 306
- "ULSQLCode enumeration" on page 445
- Object

ResetDbType method

This method is not supported in UltraLite.NET.

Syntax

Visual Basic
Public Overrides Sub **ResetDbType()**

C#
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

- "ULParameter class" on page 358
- "ULParameter members" on page 359
- "Value property" on page 372

ToString method

Returns the string representation of this instance.

Syntax

Visual Basic
Public Overrides Function **ToString()** As String

C#
public override string **ToString();**

Return value

The name of the parameter.

See also

- “ULParameter class” on page 358
- “ULParameter members” on page 359

ULParameterCollection class

Represents all parameters to a ULCommand. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULParameterCollection**
  Inherits DbParameterCollection

C#
public sealed class **ULParameterCollection**: DbParameterCollection

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.

Inherits: System.Data.Common.DbParameterCollection

Implements: System.Data.IDataParameterCollection

See also

- "ULParameterCollection members" on page 375
- "ULCommand class" on page 86
- "CommandText property" on page 93
- "Parameters property" on page 97
- DbParameterCollection
- IDataParameterCollection

ULParameterCollection members

Public properties

Member name	Description
"Count property" on page 377	Returns the number of ULParameter objects in the collection.
"IsFixedSize property" on page 377	Indicates whether the ULParameterCollection has a fixed size.

Member name	Description
"IsReadOnly property" on page 378	Indicates whether the ULParameterCollection is read-only.
"IsSynchronized property" on page 378	Indicates whether the ULParameterCollection is synchronized.
"Item properties" on page 378	Gets and sets a DbParameter in the collection.
"SyncRoot property" on page 380	Returns an object that can be used to synchronize access to the SAParameterCollection.

Public methods

Member name	Description
"Add methods" on page 380	Adds a ULParameter to the collection.
"AddRange methods" on page 386	Adds an array of values to the end of the ULParameterCollection.
"Clear method" on page 388	Removes all the parameters from the collection.
"Contains methods" on page 388	Indicates whether a DbParameter with the specified property exists in the collection.
"CopyTo method" on page 389	Copies ULParameter objects from the ULParameterCollection to the specified array.
"GetEnumerator method" on page 390	Returns an enumerator for the collection.
"IndexOf methods" on page 390	Returns the index of the specified DbParameter object.
"Insert method" on page 392	Inserts an ULParameter in the collection at the specified index.
"Remove method" on page 392	Removes an ULParameter from the collection.
"RemoveAt methods" on page 393	Removes a specified DbParameter object from the collection.

See also

- “ULParameterCollection class” on page 375
- “ULCommand class” on page 86
- “CommandText property” on page 93
- “Parameters property” on page 97
- DbParameterCollection
- IDataParameterCollection

Count property

Returns the number of ULParameter objects in the collection.

Syntax

Visual Basic
Public Overrides Readonly Property **Count** As Integer

C#
public override int **Count** { get;}

Property value

The number of ULParameter objects in the collection.

See also

- “ULParameterCollection class” on page 375
- “ULParameterCollection members” on page 375

IsFixedSize property

Indicates whether the ULParameterCollection has a fixed size.

Syntax

Visual Basic
Public Overrides Readonly Property **IsFixedSize** As Boolean

C#
public override bool **IsFixedSize** { get;}

Property value

True if this collection has a fixed size, false otherwise.

See also

- “ULParameterCollection class” on page 375
- “ULParameterCollection members” on page 375

IsReadOnly property

Indicates whether the ULParameterCollection is read-only.

Syntax

Visual Basic
Public Overrides Readonly Property **IsReadOnly** As Boolean

C#
public override bool **IsReadOnly** { get;}

Property value

True if this collection is read-only, false otherwise.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375

IsSynchronized property

Indicates whether the ULParameterCollection is synchronized.

Syntax

Visual Basic
Public Overrides Readonly Property **IsSynchronized** As Boolean

C#
public override bool **IsSynchronized** { get;}

Property value

True if this collection is synchronized, false otherwise.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375

Item properties

Gets and sets a DbParameter in the collection.

Item(Int32) property

Returns the ULParameter at the specified index. In C#, this property is the indexer for the ULParameterCollection class.

Syntax

Visual Basic

```
Public Property Item( _
  ByVal index As Integer _
) As ULParameter
```

C#

```
public 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.

Property value

The ULParameter at the specified index.

Remarks

This is the strongly-typed version of DbParameterCollection.this[int].

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "Item properties" on page 378
- "ULParameter class" on page 358
- DbParameterCollection.Item

Item(String) property

Returns the ULParameter with the specified name. In C#, this property is the indexer for the ULParameterCollection class.

Syntax

Visual Basic

```
Public Property Item( _
  ByVal parameterName As String _
) As ULParameter
```

C#

```
public ULParameter this[
  string  parameterName
] { get; set; }
```

Parameters

- **parameterName** The name of the parameter to retrieve.

Property value

The ULParameter with the specified name.

Remarks

This is the strongly-typed version of DbParameterCollection.this[string].

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "Item properties" on page 378
- "Item(Int32) property" on page 268
- "GetOrdinal method" on page 286
- "GetValue method" on page 293
- "GetFieldType method" on page 281
- "ULParameter class" on page 358

SyncRoot property

Returns an object that can be used to synchronize access to the SAParameterCollection.

Syntax

Visual Basic
Public Overrides Readonly Property **SyncRoot** As Object

C#
public override object **SyncRoot** { get;}

Property value

The object to be used to synchronize access to this collection.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375

Add methods

Adds a ULParameter to the collection.

Add(Object) method

Adds a ULParameter to the collection.

Syntax

Visual Basic

```
Public Overrides Function Add( _
  ByVal value As Object _
) As Integer
```

C#

```
public override int Add(
  object value
);
```

Parameters

- **value** The ULParameter object to add to the collection.

Return value

The index of 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

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "Add methods" on page 380
- "Add(ULParameter) method" on page 381
- "Add(String, Object) method" on page 382
- "CommandText property" on page 93
- "ULParameter class" on page 358

Add(ULParameter) method

Adds a ULParameter to the collection.

Syntax

Visual Basic

```
Public Function Add( _
  ByVal value As ULParameter _
) As ULParameter
```

C#

```
public ULParameter Add(
  ULParameter value
);
```

Parameters

- **value** The ULParameter object to add to the collection.

Return value

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

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "Add methods" on page 380
- "Add(String, Object) method" on page 382
- "ULParameter class" on page 358
- "CommandText property" on page 93

Add(String, Object) method

Adds a new ULParameter, created using the specified parameter name and value, to the collection.

Syntax

Visual Basic

```
Public Function Add( _
  ByVal parameterName As String, _
  ByVal value As Object _
) As ULParameter
```

C#

```
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.

Return value

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.

Example

The following code adds a ULParameter with the value 0 to a ULCommand called cmd.

```
' Visual Basic
cmd.Parameters.Add( "", CType( 0, Object ) )

// C#
cmd.Parameters.Add( "", (object)0 );
```

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "Add methods" on page 380
- "Add(ULParameter) method" on page 381
- "ULParameter class" on page 358
- "CommandText property" on page 93
- "ULCommand class" on page 86
- Object

Add(String, ULDbType) method

Adds a new ULParameter, created using the specified parameter name and data type, to the collection.

Syntax

Visual Basic
Public Function **Add(** _
  ByVal *parameterName* As String, _
  ByVal *ulDbType* As ULDbType _
) As ULParameter

C#
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.

Return value

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

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "Add methods" on page 380
- "Add(ULParameter) method" on page 381
- "Add(String, Object) method" on page 382
- "ULParameter class" on page 358
- "CommandText property" on page 93
- "ULCommand class" on page 86
- "ULDbType enumeration" on page 301

Add(String, ULDbType, Int32) method

Adds a new ULParameter, created using the specified parameter name, data type, and length, to the collection.

Syntax

Visual Basic

```
Public Function Add( _
  ByVal parameterName As String, _
  ByVal ulDbType As ULDbType, _
  ByVal size As Integer _
) As ULParameter
```

C#

```
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.

Return value

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

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "Add methods" on page 380
- "Add(ULParameter) method" on page 381
- "Add(String, Object) method" on page 382
- "ULParameter class" on page 358
- "CommandText property" on page 93
- "ULCommand class" on page 86
- "ULDbType enumeration" on page 301

Add(String, ULDbType, Int32, String) method

Adds a new ULParameter, created using the specified parameter name, data type, length, and source column name, to the collection.

Syntax

Visual Basic

```
Public Function Add( _
  ByVal parameterName As String, _
  ByVal ulDbType As ULDbType, _
  ByVal size As Integer, _
  ByVal sourceColumn As String _
) As ULParameter
```

C#

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.

Return value

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

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "Add methods" on page 380
- "Add(ULParameter) method" on page 381
- "Add(String, Object) method" on page 382
- "ULParameter class" on page 358
- "CommandText property" on page 93
- "ULCommand class" on page 86
- "ULDbType enumeration" on page 301

AddRange methods

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.

Syntax

Visual Basic

```
Public Overrides Sub AddRange( _
  ByVal values As Array _
)
```

C#

```
public override void AddRange(
  Array values
);
```

Parameters

- **values** An array of ULParameter objects to add to the end of this collection.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "AddRange methods" on page 386
- "ULParameter class" on page 358

AddRange(ULParameter[]) method

Adds an array of values to the end of the ULParameterCollection.

Syntax

Visual Basic

```
Public Sub AddRange( _
  ByVal values As ULParameter() _
)
```

C#

```
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

- “ULParameterCollection class” on page 375
- “ULParameterCollection members” on page 375
- “AddRange methods” on page 386
- “ULParameter class” on page 358
- DbParameterCollection.AddRange
- “ULParameterCollection class” on page 375

Clear method

Removes all the parameters from the collection.

Syntax

Visual Basic
Public Overrides Sub **Clear()**

C#
public override void **Clear();**

See also

- “ULParameterCollection class” on page 375
- “ULParameterCollection members” on page 375

Contains methods

Indicates whether a DbParameter with the specified property exists in the collection.

Contains(Object) method

Checks whether a ULParameter exists in the collection.

Syntax

Visual Basic
```
Public Overrides Function Contains( _
  ByVal value As Object _
) As Boolean
```

C#
```
public override bool Contains(
  object value
);
```

Parameters

- **value** The ULParameter object to check for.

Return value

True if the collection contains the ULParameter, false otherwise.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "Contains methods" on page 388
- "Contains(String) method" on page 389
- "ULParameter class" on page 358

Contains(String) method

Checks whether a ULParameter with the specified name exists in the collection.

Syntax

Visual Basic

```
Public Overrides Function Contains( _
  ByVal value As String _
) As Boolean
```

C#

```
public override bool Contains(
  string  value
);
```

Parameters

- **value** The name of the parameter to search for.

Return value

True if the collection contains the ULParameter, false otherwise.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "Contains methods" on page 388
- "Contains(Object) method" on page 388
- "ULParameter class" on page 358

CopyTo method

Copies ULParameter objects from the ULParameterCollection to the specified array.

Syntax

Visual Basic

```
Public Overrides Sub CopyTo( _
  ByVal array As Array, _
  ByVal index As Integer _
)
```

C#

```
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

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "ULParameter class" on page 358

GetEnumerator method

Returns an enumerator for the collection.

Syntax

Visual Basic

```
Public Overrides Function GetEnumerator() As IEnumerator
```

C#

```
public override IEnumerator GetEnumerator();
```

Return value

An ArrayList enumerator enumerating the parameters in the collection.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375

IndexOf methods

Returns the index of the specified DbParameter object.

IndexOf(Object) method

Returns the location of the ULParameter in the collection.

Syntax

Visual Basic
Public Overrides Function **IndexOf(** _
  ByVal *value* As Object _
) As Integer

C#
public override int **IndexOf(**
  object *value*
);

Parameters

- **value** The ULParameter object to locate.

Return value

The zero-based index of the ULParameter in the collection or -1 if the parameter is not found.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "IndexOf methods" on page 390
- "IndexOf(String) method" on page 391
- "ULParameter class" on page 358

IndexOf(String) method

Returns the location of the ULParameter with the specified name in the collection.

Syntax

Visual Basic
Public Overrides Function **IndexOf(** _
  ByVal *parameterName* As String _
) As Integer

C#
public override int **IndexOf(**
  string *parameterName*
);

Parameters

- **parameterName** The name of the parameter to locate.

Return value

The zero-based index of the ULParameter in the collection or -1 if the parameter is not found.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "IndexOf methods" on page 390
- "IndexOf(Object) method" on page 390
- "ULParameter class" on page 358

Insert method

Inserts an ULParameter in the collection at the specified index.

Syntax

Visual Basic
Public Overrides Sub **Insert(** _
  ByVal *index* As Integer, _
  ByVal *value* As Object _
)

C#
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.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "ULParameter class" on page 358

Remove method

Removes an ULParameter from the collection.

Syntax

Visual Basic
Public Overrides Sub **Remove(** _
  ByVal *value* As Object _
)

C#
public override void **Remove(**

object *value*
);

Parameters

- **value**   The ULParameter object to remove.

See also

- “ULParameterCollection class” on page 375
- “ULParameterCollection members” on page 375
- “ULParameter class” on page 358

RemoveAt methods

Removes a specified DbParameter object from the collection.

RemoveAt(Int32) method

Removes the parameter at the specified index in the collection.

Syntax

Visual Basic
Public Overrides Sub **RemoveAt(** _
ByVal *index* As Integer _
)

C#
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.

See also

- “ULParameterCollection class” on page 375
- “ULParameterCollection members” on page 375
- “RemoveAt methods” on page 393
- “RemoveAt(String) method” on page 393
- “Count property” on page 377

RemoveAt(String) method

Removes the parameter with the specified name from the collection.

Syntax

Visual Basic
Public Overrides Sub **RemoveAt(** _
  ByVal *parameterName* As String _
)

C#
public override void **RemoveAt(**
  string *parameterName*
);

Parameters

- **parameterName** The name of the parameter to retrieve.

See also

- "ULParameterCollection class" on page 375
- "ULParameterCollection members" on page 375
- "RemoveAt methods" on page 393
- "RemoveAt(Int32) method" on page 393

ULPublicationSchema class

UL Ext.: Represents the schema of an UltraLite publication. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULPublicationSchema**

C#
public sealed class **ULPublicationSchema**

Remarks

There is no constructor for this class. Publication schemas are created using the ULDatabaseSchema.GetPublicationSchema method of the ULDatabaseSchema class.

Some methods requiring a publication also take a comma separated list of publications.

Two special publication values are also provided by this class. ULPublicationSchema.SYNC_ALL_DB corresponds to the entire database. ULPublicationSchema.SYNC_ALL_PUBS corresponds to all publications.

See also

- "ULPublicationSchema members" on page 395
- "GetLastDownloadTime method" on page 160
- "CountUploadRows(String, UInt32) method" on page 150
- "Schema property" on page 142
- "SYNC_ALL_DB field" on page 396
- "SYNC_ALL_PUBS field" on page 396
- "GetPublicationSchema method" on page 258
- "ULDatabaseSchema class" on page 251

ULPublicationSchema members

Public fields

Member name	Description
"SYNC_ALL_DB field" on page 396	Empty publication list, corresponding to the entire database. This field is constant and read-only.
"SYNC_ALL_PUBS field" on page 396	Publication name "*", corresponding to all publications. This field is constant and read-only.

Public properties

Member name	Description
"IsOpen property" on page 397	Determines whether the publication schema is open or closed.
"Name property" on page 397	Returns the name of this publication.

See also

- "ULPublicationSchema class" on page 395
- "GetLastDownloadTime method" on page 160
- "CountUploadRows(String, UInt32) method" on page 150
- "Schema property" on page 142
- "SYNC_ALL_DB field" on page 396
- "SYNC_ALL_PUBS field" on page 396
- "GetPublicationSchema method" on page 258
- "ULDatabaseSchema class" on page 251

SYNC_ALL_DB field

Empty publication list, corresponding to the entire database. This field is constant and read-only.

Syntax

Visual Basic
Public Shared **SYNC_ALL_DB** As String

C#
public const string **SYNC_ALL_DB**;

See also

- "ULPublicationSchema class" on page 395
- "ULPublicationSchema members" on page 395

SYNC_ALL_PUBS field

Publication name "*", corresponding to all publications. This field is constant and read-only.

Syntax

Visual Basic
Public Shared **SYNC_ALL_PUBS** As String

C#
public const string **SYNC_ALL_PUBS**;

See also

- "ULPublicationSchema class" on page 395
- "ULPublicationSchema members" on page 395

IsOpen property

Determines whether the publication schema is open or closed.

Syntax

Visual Basic
Public Readonly Property **IsOpen** As Boolean

C#
public bool **IsOpen** { get;}

Property value

True if the publication schema is open, false if the publication schema closed.

See also

- "ULPublicationSchema class" on page 395
- "ULPublicationSchema members" on page 395

Name property

Returns the name of this publication.

Syntax

Visual Basic
Public Readonly Property **Name** As String

C#
public string **Name** { get;}

Property value

A string specifying the name of the publication.

See also

- "ULPublicationSchema class" on page 395
- "ULPublicationSchema members" on page 395

ULResultSet class

UL Ext.: Represents an editable result set in an UltraLite database.

Syntax

Visual Basic
Public Class **ULResultSet**
  Inherits ULDataReader

C#
public class **ULResultSet**: ULDataReader

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()

// 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.

Inherits: ULDataReader

Implements: System.Data.IDataReader, System.Data.IDataRecord, System.IDisposable

See also

- "ULResultSet members" on page 399
- "ExecuteResultSet() method" on page 115
- "ULCommand class" on page 86
- "ULResultSet class" on page 398
- "ExecuteTable() method" on page 118
- "ULDataAdapter class" on page 229
- "ULDataReader class" on page 261
- IDataReader
- IDataRecord
- IDisposable

ULResultSet members

Public properties

Member name	Description
"Depth property" on page 266 (inherited from ULDataReader)	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
"FieldCount property" on page 266 (inherited from ULDataReader)	Returns the number of columns in the cursor.
"HasRows property" on page 267 (inherited from ULDataReader)	Checks whether the ULDataReader has one or more rows.
"IsBOF property" on page 267 (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is before the first row.
"IsClosed property" on page 268 (inherited from ULDataReader)	Checks whether the cursor is currently open.
"IsEOF property" on page 268 (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is after the last row.
"Item properties" on page 268 (inherited from ULDataReader)	Gets the value of a specified column as an instance of Object.
"RecordsAffected property" on page 270 (inherited from ULDataReader)	Returns the number of rows changed, inserted, or deleted by execution of the SQL statement. For SELECT statements or CommandType.TableDirect tables, this value is -1.
"RowCount property" on page 271 (inherited from ULDataReader)	**UL Ext.:** Returns the number of rows in the cursor.
"Schema property" on page 271 (inherited from ULDataReader)	**UL Ext.:** Holds the schema of this cursor.
VisibleFieldCount (inherited from DbDataReader)	Gets the number of fields in the DbDataReader that are not hidden.

Public methods

Member name	Description
"AppendBytes method" on page 404	Appends the specified subset of the specified array of System.Bytes to the new value for the specified ULDbType.LongBinary column.
"AppendChars method" on page 406	Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType.LongVarchar column.
"Close method" on page 272 (inherited from ULDataReader)	Closes the cursor.
"Delete method" on page 407	Deletes the current row.
Dispose (inherited from DbDataReader)	Releases the resources consumed by this DbDataReader.
"GetBoolean method" on page 272 (inherited from ULDataReader)	Returns the value for the specified column as a System.Boolean.
"GetByte method" on page 273 (inherited from ULDataReader)	Returns the value for the specified column as an unsigned 8-bit value (System.Byte).
"GetBytes methods" on page 273 (inherited from ULDataReader)	**UL Ext.:** Returns the value for the specified column as an array of System.Bytes. Only valid for columns of type ULDbType.Binary, ULDbType.LongBinary, or ULDbType.UniqueIdentifier.
"GetChar method" on page 276 (inherited from ULDataReader)	This method is not supported in UltraLite.NET.
"GetChars method" on page 277 (inherited from ULDataReader)	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 (inherited from DbDataReader)	Returns a DbDataReader object for the requested column ordinal.
"GetDataTypeName method" on page 278 (inherited from ULDataReader)	Returns the name of the specified column's provider data type.
"GetDateTime method" on page 279 (inherited from ULDataReader)	Returns the value for the specified column as a System.DateTime with millisecond accuracy.

Member name	**Description**
"GetDecimal method" on page 279 (inherited from ULDataReader)	Returns the value for the specified column as a System.Decimal.
"GetDouble method" on page 280 (inherited from ULDataReader)	Returns the value for the specified column as a System.Double.
"GetEnumerator method" on page 281 (inherited from ULDataReader)	Returns an System.Collections.IEnumerator that iterates through the ULDataReader.
"GetFieldType method" on page 281 (inherited from ULDataReader)	Returns the System.Type most appropriate for the specified column.
"GetFloat method" on page 282 (inherited from ULDataReader)	Returns the value for the specified column as a System.Single.
"GetGuid method" on page 282 (inherited from ULDataReader)	Returns the value for the specified column as a UUID (System.Guid).
"GetInt16 method" on page 283 (inherited from ULDataReader)	Returns the value for the specified column as a System.Int16.
"GetInt32 method" on page 284 (inherited from ULDataReader)	Returns the value for the specified column as an Int32.
"GetInt64 method" on page 285 (inherited from ULDataReader)	Returns the value for the specified column as an Int64.
"GetName method" on page 285 (inherited from ULDataReader)	Returns the name of the specified column.
"GetOrdinal method" on page 286 (inherited from ULDataReader)	Returns the column ID of the named column.
GetProviderSpecificFieldType (inherited from DbDataReader)	Returns the provider-specific field type of the specified column.
GetProviderSpecificValue (inherited from DbDataReader)	Gets the value of the specified column as an instance of Object.
GetProviderSpecificValues (inherited from DbDataReader)	Gets all provider-specific attribute columns in the collection for the current row.

Member name	Description
"GetSchemaTable method" on page 288 (inherited from ULDataReader)	Returns a System.Data.DataTable that describes the column metadata of the ULDataReader.
"GetString method" on page 290 (inherited from ULDataReader)	Returns the value for the specified column as a System.String.
"GetTimeSpan method" on page 290 (inherited from ULDataReader)	Returns the value for the specified column as a System.TimeSpan with millisecond accuracy.
"GetUInt16 method" on page 291 (inherited from ULDataReader)	Returns the value for the specified column as a System.UInt16.
"GetUInt32 method" on page 292 (inherited from ULDataReader)	Returns the value for the specified column as a UInt32.
"GetUInt64 method" on page 292 (inherited from ULDataReader)	Returns the value for the specified column as a System.UInt64.
"GetValue method" on page 293 (inherited from ULDataReader)	Returns the value of the specified column in its native format.
"GetValues method" on page 294 (inherited from ULDataReader)	Returns all the column values for the current row.
"IsDBNull method" on page 295 (inherited from ULDataReader)	Checks whether the value from the specified column is NULL.
"MoveAfterLast method" on page 295 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to after the last row of the cursor.
"MoveBeforeFirst method" on page 296 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to before the first row of the cursor.
"MoveFirst method" on page 296 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the first row of the cursor.

Member name	Description
“MoveLast method” on page 296 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the last row of the cursor.
“MoveNext method” on page 297 (inherited from ULDataReader)	**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” on page 297 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the previous row or before the first row.
“MoveRelative method” on page 298 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor relative to the current row.
“NextResult method” on page 299 (inherited from ULDataReader)	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
“Read method” on page 299 (inherited from ULDataReader)	Positions the cursor to the next row, or after the last row if the cursor was already on the last row.
“SetBoolean method” on page 408	Sets the value for the specified column using a System.Boolean.
“SetByte method” on page 409	Sets the value for the specified column using a System.Byte (unsigned 8-bit integer).
“SetBytes method” on page 409	Sets the value for the specified column using an array of System.Bytes.
“SetDBNull method” on page 410	Sets a column to NULL.
“SetDateTime method” on page 411	Sets the value for the specified column using a ="System.DateTime.
“SetDecimal method” on page 412	Sets the value for the specified column using a System.Decimal.
“SetDouble method” on page 413	Sets the value for the specified column using a System.Double.
“SetFloat method” on page 414	Sets the value for the specified column using a System.Single.
“SetGuid method” on page 415	Sets the value for the specified column using a System.Guid.

Member name	Description
"SetInt16 method" on page 416	Sets the value for the specified column using an System.Int16.
"SetInt32 method" on page 417	Sets the value for the specified column using an System.Int32.
"SetInt64 method" on page 418	Sets the value for the specified column using an Int64.
"SetString method" on page 419	Sets the value for the specified column using a System.String.
"SetTimeSpan method" on page 420	Sets the value for the specified column using a System.TimeSpan.
"SetToDefault method" on page 420	Sets the value for the specified column to its default value.
"SetUInt16 method" on page 421	Sets the value for the specified column using a System.UInt16.
"SetUInt32 method" on page 422	Sets the value for the specified column using an System.UInt32.
"SetUInt64 method" on page 423	Sets the value for the specified column using a System.UInt64.
"Update method" on page 424	Updates the current row with the current column values (specified using the set methods).

See also

- "ULResultSet class" on page 398
- "ExecuteResultSet() method" on page 115
- "ULCommand class" on page 86
- "ULResultSet class" on page 398
- "ExecuteTable() method" on page 118
- "ULDataAdapter class" on page 229
- "ULDataReader class" on page 261
- IDataReader
- IDataRecord
- IDisposable

AppendBytes method

Appends the specified subset of the specified array of System.Bytes to the new value for the specified ULDbType.LongBinary column.

Syntax

Visual Basic

```
Public Sub AppendBytes( _
   ByVal columnID As Integer, _
   ByVal val As Byte(), _
   ByVal srcOffset As Integer, _
   ByVal count As Integer _
)
```

C#

```
public void AppendBytes(
   int columnID,
   byte[] val,
   int srcOffset,
   int count
);
```

Parameters

- **columnID** 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.

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 *val*.Length.

For other errors, a ULException with the appropriate error code is thrown.

See also

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- Byte
- "ULDbType enumeration" on page 301
- "InsertBegin method" on page 536
- "Insert method" on page 536
- "ULException class" on page 306
- "ULSQLCode enumeration" on page 445
- "ULException class" on page 306
- "FieldCount property" on page 266

AppendChars method

Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType.LongVarchar column.

Syntax

Visual Basic

```
Public Sub AppendChars( _
  ByVal columnID As Integer, _
  ByVal val As Char(), _
  ByVal srcOffset As Integer, _
  ByVal count As Integer _
)
```

C#

```
public void AppendChars(
  int columnID,
  char[] val,
  int srcOffset,
  int count
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- Char
- "ULDbType enumeration" on page 301
- "InsertBegin method" on page 536
- "Insert method" on page 536
- "ULException class" on page 306
- "ULSQLCode enumeration" on page 445
- "FieldCount property" on page 266

Delete method

Deletes the current row.

Syntax

Visual Basic
Public Sub **Delete()**

C#
public void **Delete();**

See also

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "StartSynchronizationDelete method" on page 171
- "StopSynchronizationDelete method" on page 172

SetBoolean method

Sets the value for the specified column using a System.Boolean.

Syntax

Visual Basic

```
Public Sub SetBoolean( _
  ByVal columnID As Integer, _
  ByVal val As Boolean _
)
```

C#

```
public void SetBoolean(
  int columnID,
  bool val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- Boolean
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetByte method

Sets the value for the specified column using a System.Byte (unsigned 8-bit integer).

Syntax

Visual Basic

```
Public Sub SetByte( _
  ByVal columnID As Integer, _
  ByVal val As Byte _
)
```

C#

```
public void SetByte(
  int columnID,
  byte val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- Byte
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetBytes method

Sets the value for the specified column using an array of System.Bytes.

Syntax

Visual Basic

```
Public Sub SetBytes( _
  ByVal columnID As Integer, _
  ByVal val As Byte() _
)
```

C#
```
public void SetBytes(
  int columnID,
  byte[] val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- Byte
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266
- "ULDbType enumeration" on page 301
- "ULDbType enumeration" on page 301
- "ULDbType enumeration" on page 301

SetDBNull method

Sets a column to NULL.

Syntax

Visual Basic
```
Public Sub SetDBNull( _
  ByVal columnID As Integer _
)
```

C#
```
public void SetDBNull(
  int columnID
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "IsColumnNullable method" on page 554
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetDateTime method

Sets the value for the specified column using a System.DateTime.

Syntax

Visual Basic
Public Sub **SetDateTime(** _
  ByVal *columnID* As Integer, _
  ByVal *val* As Date _
)

C#
public void **SetDateTime(**
  int *columnID*,
  DateTime *val*
);

Parameters

- **columnID** 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.

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

- “ULResultSet class” on page 398
- “ULResultSet members” on page 399
- “GetOrdinal method” on page 286
- “Schema property” on page 271
- “GetFieldType method” on page 281
- DateTime
- “Insert method” on page 536
- “Update method” on page 424
- “FieldCount property” on page 266

SetDecimal method

Sets the value for the specified column using a System.Decimal.

Syntax

Visual Basic

```
Public Sub SetDecimal( _
  ByVal columnID As Integer, _
  ByVal val As Decimal _
)
```

C#

```
public void SetDecimal(
  int columnID,
  decimal val
);
```

Parameters

- **columnID** 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.

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

- “ULResultSet class” on page 398
- “ULResultSet members” on page 399
- “GetOrdinal method” on page 286
- “Schema property” on page 271
- “GetFieldType method” on page 281
- Decimal
- “Insert method” on page 536
- “Update method” on page 424
- “FieldCount property” on page 266

SetDouble method

Sets the value for the specified column using a System.Double.

Syntax

Visual Basic

```
Public Sub SetDouble( _
  ByVal columnID As Integer, _
  ByVal val As Double _
)
```

C#

```
public void SetDouble(
  int columnID,
  double val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- Double
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetFloat method

Sets the value for the specified column using a System.Single.

Syntax

Visual Basic

```
Public Sub SetFloat( _
  ByVal columnID As Integer, _
  ByVal val As Single _
)
```

C#

```
public void SetFloat(
  int columnID,
  float  val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- Single
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetGuid method

Sets the value for the specified column using a System.Guid.

Syntax

Visual Basic

```
Public Sub SetGuid( _
  ByVal columnID As Integer, _
  ByVal val As Guid _
)
```

C#

```
public void SetGuid(
  int columnID,
  Guid val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetNewUUID method" on page 161
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- "GetColumnSize method" on page 226
- Guid
- "Insert method" on page 536
- "Update method" on page 424
- "ULDbType enumeration" on page 301
- "ULDbType enumeration" on page 301
- "FieldCount property" on page 266

SetInt16 method

Sets the value for the specified column using an System.Int16.

Syntax

Visual Basic

```
Public Sub SetInt16( _
   ByVal columnID As Integer, _
   ByVal val As Short _
)
```

C#

```
public void SetInt16(
   int columnID,
   short val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- Int16
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetInt32 method

Sets the value for the specified column using an System.Int32.

Syntax

Visual Basic

```
Public Sub SetInt32( _
  ByVal columnID As Integer, _
  ByVal val As Integer _
)
```

C#

```
public void SetInt32(
  int columnID,
  int val
);
```

Parameters

- **columnID** 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.

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

- “ULResultSet class” on page 398
- “ULResultSet members” on page 399
- “GetOrdinal method” on page 286
- “Schema property” on page 271
- “GetFieldType method” on page 281
- Int32
- “Insert method” on page 536
- “Update method” on page 424
- “FieldCount property” on page 266

SetInt64 method

Sets the value for the specified column using an Int64.

Syntax

Visual Basic

```
Public Sub SetInt64( _
  ByVal columnID As Integer, _
  ByVal val As Long _
)
```

C#

```
public void SetInt64(
  int columnID,
  long val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- Int64
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetString method

Sets the value for the specified column using a System.String.

Syntax

Visual Basic

```
Public Sub SetString( _
  ByVal columnID As Integer, _
  ByVal val As String _
)
```

C#

```
public void SetString(
  int columnID,
  string  val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetTimeSpan method

Sets the value for the specified column using a System.TimeSpan.

Syntax

Visual Basic

```
Public Sub SetTimeSpan( _
  ByVal columnID As Integer, _
  ByVal val As TimeSpan _
)
```

C#

```
public void SetTimeSpan(
  int columnID,
  TimeSpan val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- TimeSpan
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetToDefault method

Sets the value for the specified column to its default value.

Syntax

Visual Basic

```
Public Sub SetToDefault( _
  ByVal columnID As Integer _
)
```

C#

```
public void SetToDefault(
  int columnID
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "Schema property" on page 271
- "GetFieldType method" on page 281
- "GetColumnDefaultValue method" on page 547
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetUInt16 method

Sets the value for the specified column using a System.UInt16.

Syntax

Visual Basic

```
Public Sub SetUInt16( _
  ByVal columnID As Integer, _
  ByVal val As UInt16 _
)
```

C#

```
public void SetUInt16(
  int columnID,
  ushort val
);
```

Parameters

- **columnID** 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.

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

- "ULResultSet class" on page 398
- "ULResultSet members" on page 399
- "GetOrdinal method" on page 286
- "GetFieldType method" on page 281
- UInt16
- "Insert method" on page 536
- "Update method" on page 424
- "FieldCount property" on page 266

SetUInt32 method

Sets the value for the specified column using an System.UInt32.

Syntax

Visual Basic

```
Public Sub SetUInt32( _
  ByVal columnID As Integer, _
  ByVal val As UInt32 _
)
```

C#

```
public void SetUInt32(
  int columnID,
  uint val
);
```

Parameters

- **columnID** 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.

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

- “ULResultSet class” on page 398
- “ULResultSet members” on page 399
- “GetOrdinal method” on page 286
- “Schema property” on page 271
- “GetFieldType method” on page 281
- UInt32
- “Insert method” on page 536
- “Update method” on page 424
- “FieldCount property” on page 266

SetUInt64 method

Sets the value for the specified column using a System.UInt64.

Syntax

Visual Basic

```
Public Sub SetUInt64( _
  ByVal columnID As Integer, _
  ByVal val As UInt64 _
)
```

C#

```
public void SetUInt64(
  int columnID,
  ulong val
);
```

Parameters

- **columnID** 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.

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

- “ULResultSet class” on page 398
- “ULResultSet members” on page 399
- “GetOrdinal method” on page 286
- “Schema property” on page 271
- “GetFieldType method” on page 281
- UInt64
- “Insert method” on page 536
- “Update method” on page 424
- “FieldCount property” on page 266

Update method

Updates the current row with the current column values (specified using the set methods).

Syntax

Visual Basic
Public Sub **Update()**

C#
public void **Update();**

See also

- “ULResultSet class” on page 398
- “ULResultSet members” on page 399

ULResultSetSchema class

UL Ext.: Represents the schema of an UltraLite result set. This class cannot be inherited.

Syntax

Visual Basic

```
Public NotInheritable Class ULResultSetSchema
  Inherits ULCursorSchema
```

C#

```
public sealed class ULResultSetSchema: ULCursorSchema
```

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.

Inherits: ULCursorSchema

See also

- "ULResultSetSchema members" on page 425
- "ULCommand class" on page 86
- "ULDataReader class" on page 261
- "ULResultSetSchema class" on page 425
- "Schema property" on page 271
- "ULCursorSchema class" on page 220

ULResultSetSchema members

Public properties

Member name	Description
"ColumnCount property" on page 221 (inherited from ULCursorSchema)	Returns the number of columns in the cursor.
"IsOpen property" on page 222 (inherited from ULCursorSchema)	Checks whether the cursor schema is currently open.
"Name property" on page 426	Returns the name of the cursor.

Public methods

Member name	Description
"GetColumnID method" on page 222 (inherited from ULCursorSchema)	Returns the column ID of the named column.
"GetColumnName method" on page 223 (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
"GetColumnPrecision method" on page 224 (inherited from ULCursorSchema)	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnSQLName method" on page 225 (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
"GetColumnScale method" on page 226 (inherited from ULCursorSchema)	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnSize method" on page 226 (inherited from ULCursorSchema)	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
"GetColumnULDbType method" on page 227 (inherited from ULCursorSchema)	Returns the UltraLite.NET data type of the column identified by the specified column ID.
"GetSchemaTable method" on page 227 (inherited from ULCursorSchema)	Returns a System.Data.DataTable that describes the column schema of the ULDataReader.

See also

- "ULResultSetSchema class" on page 425
- "ULCommand class" on page 86
- "ULDataReader class" on page 261
- "ULResultSetSchema class" on page 425
- "Schema property" on page 271
- "ULCursorSchema class" on page 220

Name property

Returns the name of the cursor.

Syntax

Visual Basic
Public Overrides Readonly Property **Name** As String

C#
public override string **Name** { get;}

Property value

The SQL statement that generated the ULResultSetSchema.

See also

- "ULResultSetSchema class" on page 425
- "ULResultSetSchema members" on page 425

ULRowsCopiedEventArgs class

Represents the set of arguments passed to the ULRowsCopiedEventHandler. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULRowsCopiedEventArgs**

C#
public sealed class **ULRowsCopiedEventArgs**

Remarks

Restrictions: The ULRowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

See also

- "ULRowsCopiedEventArgs members" on page 428
- "ULRowsCopiedEventHandler delegate" on page 431

ULRowsCopiedEventArgs members

Public constructors

Member name	Description
"ULRowsCopiedEventArgs constructor" on page 428	Creates a new instance of the ULRowsCopiedEventArgs object.

Public properties

Member name	Description
"Abort property" on page 429	Gets or sets a value that indicates whether the bulk-copy operation should be aborted.
"RowsCopied property" on page 429	Returns the number of rows copied during the current bulk-copy operation.

See also

- "ULRowsCopiedEventArgs class" on page 428
- "ULRowsCopiedEventHandler delegate" on page 431

ULRowsCopiedEventArgs constructor

Creates a new instance of the ULRowsCopiedEventArgs object.

Syntax

Visual Basic
```
Public Sub New( _
  ByVal rowsCopied As Long _
)
```

C#
```
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

Restrictions: The ULRowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

See also

- "ULRowsCopiedEventArgs class" on page 428
- "ULRowsCopiedEventArgs members" on page 428

Abort property

Gets or sets a value that indicates whether the bulk-copy operation should be aborted.

Syntax

Visual Basic
```
Public Property Abort As Boolean
```

C#
```
public bool Abort { get; set; }
```

Remarks

Restrictions: The ULRowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

See also

- "ULRowsCopiedEventArgs class" on page 428
- "ULRowsCopiedEventArgs members" on page 428

RowsCopied property

Returns the number of rows copied during the current bulk-copy operation.

Syntax

Visual Basic
Public Readonly Property **RowsCopied** As Long

C#
public long **RowsCopied** { get;}

Property value

A long integer representing the number of rows copied.

Remarks

Restrictions: The ULRowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

See also

- "ULRowsCopiedEventArgs class" on page 428
- "ULRowsCopiedEventArgs members" on page 428

ULRowsCopiedEventHandler delegate

Represents the method that will handle the ULBulkCopy.ULRowsCopied event.

Syntax

Visual Basic

```
Public Delegate Sub ULRowsCopiedEventHandler( _
  ByVal sender As Object, _
  ByVal rowsCopiedEventArgs As ULRowsCopiedEventArgs _
)
```

C#

```
public delegate void ULRowsCopiedEventHandler(
  object sender,
  ULRowsCopiedEventArgs rowsCopiedEventArgs
);
```

Remarks

Restrictions: The ULRowsCopiedEventHandler delegate is not available in the .NET Compact Framework 2.0.

See also

- "ULRowsCopied event" on page 67
- Object

ULRowUpdatedEventArgs class

Provides data for the ULDataAdapter.RowUpdated event. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULRowUpdatedEventArgs**
  Inherits RowUpdatedEventArgs

C#
public sealed class **ULRowUpdatedEventArgs**: RowUpdatedEventArgs

Remarks

Inherits: System.Data.Common.RowUpdatedEventArgs

See also

- "ULRowUpdatedEventArgs members" on page 432
- "RowUpdated event" on page 239
- RowUpdatedEventArgs

ULRowUpdatedEventArgs members

Public constructors

Member name	Description
"ULRowUpdatedEventArgs constructor" on page 433	Initializes a new instance of the ULRowUpdatedEventArgs class.

Public properties

Member name	Description
"Command property" on page 434	Returns the ULCommand executed when DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping) is called.
Errors (inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the RowUpdatedEventArgs.Command was executed.
"RecordsAffected property" on page 435	Returns the number of rows changed, inserted, or deleted by the execution of the SQL statement. For SELECT statements this value is -1.
Row (inherited from RowUpdatedEventArgs)	Gets the DataRow sent through an DbDataAdapter.Update.

Member name	Description
RowCount (inherited from RowUpdatedEventArgs)	Gets the number of rows processed in a batch of updated records.
StatementType (inherited from RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status (inherited from RowUpdatedEventArgs)	Gets the UpdateStatus of the RowUpdatedEventArgs.Command.
TableMapping (inherited from RowUpdatedEventArgs)	Gets the DataTableMapping sent through an DbDataAdapter.Update.

Public methods

Member name	Description
CopyToRows (inherited from RowUpdatedEventArgs)	Lets you access the rows processed during a batch update operation.

See also

- "ULRowUpdatedEventArgs class" on page 432
- "RowUpdated event" on page 239
- RowUpdatedEventArgs

ULRowUpdatedEventArgs constructor

Initializes a new instance of the ULRowUpdatedEventArgs class.

Syntax

Visual Basic

```
Public Sub New( _
  ByVal row As DataRow, _
  ByVal command As IDbCommand, _
  ByVal statementType As StatementType, _
  ByVal tableMapping As DataTableMapping _
)
```

C#

```
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

- "ULRowUpdatedEventArgs class" on page 432
- "ULRowUpdatedEventArgs members" on page 432
- DataRow
- IDbCommand
- StatementType
- DataTableMapping
- DbDataAdapter.Update

Command property

Returns the ULCommand executed when DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping) is called.

Syntax

Visual Basic
Public Readonly Property **Command** As ULCommand

C#
public ULCommand **Command** { get;}

Property value

The ULCommand object executed by the update.

Remarks

This is the strongly-typed version of System.Data.Common.RowUpdatedEventArgs.Command.

See also

- "ULRowUpdatedEventArgs class" on page 432
- "ULRowUpdatedEventArgs members" on page 432
- "ULCommand class" on page 86
- DbDataAdapter.Update
- RowUpdatedEventArgs.Command

RecordsAffected property

Returns the number of rows changed, inserted, or deleted by the execution of the SQL statement. For SELECT statements this value is -1.

Syntax

Visual Basic
Public Readonly Property **RecordsAffected** As Integer

C#
public int **RecordsAffected** { get;}

Property value

The number of rows changed, inserted, or deleted; 0 if no rows were affected or the statement failed; and -1 for SELECT statements.

See also

- "ULRowUpdatedEventArgs class" on page 432
- "ULRowUpdatedEventArgs members" on page 432

ULRowUpdatedEventHandler delegate

Represents the method that will handle the ULDataAdapter.RowUpdated event.

Syntax

Visual Basic

```
Public Delegate Sub ULRowUpdatedEventHandler( _
   ByVal sender As Object, _
   ByVal e As ULRowUpdatedEventArgs _
)
```

C#

```
public delegate void ULRowUpdatedEventHandler(
   object sender,
   ULRowUpdatedEventArgs e
);
```

See also

- "RowUpdated event" on page 239
- "ULRowUpdatedEventArgs class" on page 432

ULRowUpdatingEventArgs class

Provides data for the ULDataAdapter.RowUpdating event. This class cannot be inherited.

Syntax

Visual Basic

```
Public NotInheritable Class ULRowUpdatingEventArgs
  Inherits RowUpdatingEventArgs
```

C#

```
public sealed class ULRowUpdatingEventArgs: RowUpdatingEventArgs
```

Remarks

Inherits: System.Data.Common.RowUpdatingEventArgs

See also

- "ULRowUpdatingEventArgs members" on page 437
- "RowUpdating event" on page 240
- RowUpdatingEventArgs

ULRowUpdatingEventArgs members

Public constructors

Member name	Description
"ULRowUpdatingEventArgs constructor" on page 438	Initializes a new instance of the ULRowUpdatingEventArgs class.

Public properties

Member name	Description
"Command property" on page 439	Specifies the ULCommand to execute when performing the DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping).
Errors (inherited from RowUpdatingEventArgs)	Gets any errors generated by the .NET Framework data provider when the RowUpdatedEventArgs.Command executes.
Row (inherited from RowUpdatingEventArgs)	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType (inherited from RowUpdatingEventArgs)	Gets the type of SQL statement to execute.

Member name	Description
Status (inherited from RowUpdatingEventArgs)	Gets or sets the UpdateStatus of the RowUpdatedEventArgs.Command.
TableMapping (inherited from RowUpdatingEventArgs)	Gets the DataTableMapping to send through the DbDataAdapter.Update.

See also

- "ULRowUpdatingEventArgs class" on page 437
- "RowUpdating event" on page 240
- RowUpdatingEventArgs

ULRowUpdatingEventArgs constructor

Initializes a new instance of the ULRowUpdatingEventArgs class.

Syntax

Visual Basic

```
Public Sub New( _
  ByVal row As DataRow, _
  ByVal command As IDbCommand, _
  ByVal statementType As StatementType, _
  ByVal tableMapping As DataTableMapping _
)
```

C#

```
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

- "ULRowUpdatingEventArgs class" on page 437
- "ULRowUpdatingEventArgs members" on page 437
- DataRow
- IDbCommand
- StatementType
- DataTableMapping
- DbDataAdapter.Update

Command property

Specifies the ULCommand to execute when performing the DbDataAdapter.Update(System.Data.DataRow[],System.Data.Common.DataTableMapping).

Syntax

Visual Basic
Public Property **Command** As ULCommand

C#
public ULCommand **Command** { get; set; }

Property value

The ULCommand object to execute when updating.

Remarks

This is the strongly-typed version of System.Data.Common.RowUpdatingEventArgs.Command.

See also

- "ULRowUpdatingEventArgs class" on page 437
- "ULRowUpdatingEventArgs members" on page 437
- "ULCommand class" on page 86
- DbDataAdapter.Update
- RowUpdatingEventArgs.Command

ULRowUpdatingEventHandler delegate

Represents the method that will handle the ULDataAdapter.RowUpdating event.

Syntax

Visual Basic

```
Public Delegate Sub ULRowUpdatingEventHandler( _
  ByVal sender As Object, _
  ByVal e As ULRowUpdatingEventArgs _
)
```

C#

```
public delegate void ULRowUpdatingEventHandler(
  object sender,
  ULRowUpdatingEventArgs e
);
```

See also

- "RowUpdating event" on page 240
- "ULRowUpdatingEventArgs class" on page 437

ULRuntimeType enumeration

UL Ext.: Enumerates the types of UltraLite.NET runtimes.

Syntax

Visual Basic
Public Enum **ULRuntimeType**

C#
public enum **ULRuntimeType**

Members

Member name	Description	Value
STANDALONE_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_ENGINE_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 ap-plications.	1

See also

- "RuntimeType property" on page 243

ULServerSyncListener interface

UL Ext.: The listener interface for receiving server synchronization messages.

Syntax

Visual Basic
Public Interface **ULServerSyncListener**

C#
public interface **ULServerSyncListener**

See also

- "ULServerSyncListener members" on page 442

ULServerSyncListener members

Public methods

Member name	Description
"ServerSyncInvoked method" on page 442	Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

See also

- "ULServerSyncListener interface" on page 442

ServerSyncInvoked method

Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

Syntax

Visual Basic
```
Public Sub ServerSyncInvoked( _
  ByVal messageName As String _
)
```

C#
```
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 code fragments demonstrate how to receive a server synchronization request and perform a synchronization in the UI thread.

```
' Visual Basic
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


// C#
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(
              "myCompnay.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();
        }
      }
```

See also

- "ULServerSyncListener interface" on page 442
- "ULServerSyncListener members" on page 442

ULSQLCode enumeration

UL Ext.: Enumerates the SQL codes that may be reported by UltraLite.NET.

Syntax

Visual Basic
Public Enum **ULSQLCode**

C#
public enum **ULSQLCode**

Members

Member name	Description	Value
SQLE_AGGREGATES_NOT_ALLOWED	See "Invalid use of an aggregate function" [*Error Messages*].	-150
SQLE_ALIAS_NOT_UNIQUE	See "Alias '%1' is not unique" [*Error Messages*].	-830
SQLE_ALIAS_NOT_YET_DEFINED	See "Definition for alias '%1' must appear before its first reference" [*Error Messages*].	-831
SQLE_AMBIGUOUS_INDEX_NAME	See "Index name '%1' is ambiguous" [*Error Messages*].	-678
SQLE_ARGUMENT_CANNOT_BE_NULL	See "Argument %1 of procedure '%2' cannot be NULL" [*Error Messages*].	-90
SQLE_BAD_ENCRYPTION_KEY	See "Incorrect or missing encryption key" [*Error Messages*].	-840
SQLE_BAD_PARAM_INDEX	See "Input parameter index out of range" [*Error Messages*].	-689
SQLE_CANNOT_ACCESS_FILE	See "Cannot access file '%1' -- %2" [*Error Messages*].	-602
SQLE_CANNOT_ACCESS_FILESYSTEM	See "Unable to access the filesystem on the device" [*Error Messages*].	-1108
SQLE_CANNOT_ACCESS_SCHEMA_FILE	See "Cannot access schema file '%1'" [*Error Messages*].	-951
SQLE_CANNOT_CHANGE_ML_REMOTE_ID	See "Cannot change the MobiLink remote id when the status of the last upload is unknown" [*Error Messages*].	-1118

Member name	**Description**	**Value**
SQLE_CANNOT_EXECUTE_STMT	See "Statement cannot be executed" [*Error Messages*].	111
SQLE_CANNOT_MODIFY	See "Cannot modify column '%1' in table '%2'" [*Error Messages*].	-191
SQLE_CANNOT_OPEN_LOG	See "Cannot open transaction log file -- %1" [*Error Messages*].	-106
SQLE_CANNOT_REGISTER_LISTENER	See "The specified listener could not be registered" [*Error Messages*].	-992
SQLE_CLIENT_OUT_OF_MEMORY	See "Client out of memory" [*Error Messages*].	-876
SQLE_COLUMN_AMBIGUOUS	See "Column '%1' found in more than one table -- need a correlation name" [*Error Messages*].	-144
SQLE_COLUMN_CANNOT_BE_NULL	See "Column '%1' in table '%2' cannot be NULL" [*Error Messages*].	-195
SQLE_COLUMN_IN_INDEX	See "Cannot alter a column in an index" [*Error Messages*].	-127
SQLE_COLUMN_NOT_FOUND	See "Column '%1' not found" [*Error Messages*].	-143
SQLE_COLUMN_NOT_INDEXED	See "Column '%1' not part of any indexes in its containing table" [*Error Messages*].	-1101
SQLE_COLUMN_NOT_STREAMABLE	See "The operation failed because column '%1''s type does not support streaming" [*Error Messages*].	-1100
SQLE_COMMUNICATIONS_ERROR	See "Communication error" [*Error Messages*].	-85
SQLE_CONNECTION_ALREADY_EXISTS	See "This connection already exists" [*Error Messages*].	-955
SQLE_CONNECTION_NOT_FOUND	See "Connection not found" [*Error Messages*].	-108
SQLE_CONNECTION_RESTORED	See "UltraLite connection was restored" [*Error Messages*].	133
SQLE_CONSTRAINT_NOT_FOUND	See "Constraint '%1' not found" [*Error Messages*].	-929
SQLE_CONVERSION_ERROR	See "Cannot convert %1 to a %2" [*Error Messages*].	-157

Member name	Description	Value
SQLE_CORRUPT_PAGE_READ_RETRY	See "Retrying read of corrupt page (page '%1')" [*Error Messages*].	143
SQLE_CORRUPT_ULTRALITE_DATABASE	See "Database page validation failed with code: %1" [*Error Messages*].	-1186
SQLE_CORRUPT_ULTRALITE_INDEX	See "Index validation failed for table %1, index %2 with code: %3" [*Error Messages*].	-1185
SQLE_COULD_NOT_FIND_FUNCTION	See "Could not find '%1' in dynamic library '%2'" [*Error Messages*].	-621
SQLE_COULD_NOT_LOAD_LIBRARY	See "Could not load dynamic library '%1'" [*Error Messages*].	-620
SQLE_CURSOR_ALREADY_OPEN	See "Cursor already open" [*Error Messages*].	-172
SQLE_CURSOR_NOT_DECLARED	See "Cursor has not been declared" [*Error Messages*].	-170
SQLE_CURSOR_NOT_OPEN	See "Cursor not open" [*Error Messages*].	-180
SQLE_CURSOR_RESTORED	See "UltraLite cursor (or result set or table) was restored" [*Error Messages*].	134
SQLE_CURSOROP_NOT_ALLOWED	See "Illegal cursor operation attempt" [*Error Messages*].	-187
SQLE_DATABASE_ERROR	See "Internal database error %1 -- transaction rolled back" [*Error Messages*].	-301
SQLE_DATABASE_NAME_REQUIRED	See "Database name required to start server" [*Error Messages*].	-87
SQLE_DATABASE_NOT_CREATED	See "Database creation failed: %1" [*Error Messages*].	-645
SQLE_DATABASE_STATE_RESTORED	See "UltraLite database state was restored" [*Error Messages*].	142
SQLE_DATABASE_UPGRADE_WARNING	See "Automatic database upgrade applied" [*Error Messages*].	149
SQLE_DATATYPE_NOT_ALLOWED	See "Expression has unsupported data type" [*Error Messages*].	-624

Member name	Description	Value
SQLE_DBSPACE_FULL	See "A dbspace has reached its maximum file size" [*Error Messages*].	-604
SQLE_DESCRIBE_NONSELECT	See "Can only describe a SELECT statement" [*Error Messages*].	-160
SQLE_DEVICE_ERROR	See "I/O error %1 -- transaction rolled back" [*Error Messages*].	-305
SQLE_DEVICE_IO_FAILED	See "File I/O failed for '%1'" [*Error Messages*].	-974
SQLE_DIV_ZERO_ERROR	See "Division by zero" [*Error Messages*].	-628
SQLE_DOWNLOAD_CONFLICT	See "Download failed because of conflicts with existing rows" [*Error Messages*].	-839
SQLE_DOWNLOAD_RESTART_FAILED	See "Unable to retry download because upload is not finished" [*Error Messages*].	-1102
SQLE_DROP_DATABASE_FAILED	See "An attempt to delete database '%1' failed" [*Error Messages*].	-651
SQLE_DUPLICATE_CURSOR_NAME	See "The cursor name '%1' already exists" [*Error Messages*].	-683
SQLE_DUPLICATE_FOREIGN_KEY	See "Foreign key '%1' for table '%2' duplicates an existing foreign key" [*Error Messages*].	-251
SQLE_DUPLICATE_OPTION	See "Option '%1' specified more than once" [*Error Messages*].	139
SQLE_DUPLICATE_ROW_FOUND_IN_DOWNLOAD	See "Two rows with the same primary key have been download for table '%1'" [*Error Messages*].	145
SQLE_DYNAMIC_MEMORY_EXHAUSTED	See "Dynamic memory exhausted" [*Error Messages*].	-78
SQLE_ENCRYPTION_INITIALIZATION_FAILED	See "Could not initialize the encryption DLL: '%1'" [*Error Messages*].	-984
SQLE_ENCRYPTION_NOT_ENABLED	See "Encryption has not been enabled" [*Error Messages*].	-1143
SQLE_ENCRYPTION_NOT_ENABLED_WARNING	See "Encryption has not been enabled" [*Error Messages*].	140

Member name	Description	Value
SQLE_ENGINE_ALREADY_RUNNING	See "Database server already running" [*Error Messages*].	-96
SQLE_ERROR	See "Run time SQL error -- %1" [*Error Messages*].	-300
SQLE_ERROR_CALLING_FUNCTION	See "Could not allocate resources to call external function" [*Error Messages*].	-622
SQLE_ERROR_IN_ASSIGNMENT	See "Error in assignment" [*Error Messages*].	-641
SQLE_EVENT_NOT_FOUND	See "Event '%1' not found" [*Error Messages*].	-771
SQLE_EVENT_NOTIFICATION_QUEUE_FULL	See "Event notification queue '%1' is full, notification discarded" [*Error Messages*].	146
SQLE_EVENT_NOTIFICATION_QUEUE_NOT_FOUND	See "Event notification queue '%1' not found" [*Error Messages*].	-1263
SQLE_EVENT_NOTIFICATION_QUEUE_NOT_FOUND_WARN	See "Event notification queue '%1' not found warning" [*Error Messages*].	148
SQLE_EVENT_NOTIFICATION_QUEUE_TIMEOUT	See "No notification within timeout on queue '%1'" [*Error Messages*].	-1266
SQLE_EVENT_NOTIFICATIONS_LOST	See "Event notifications lost on queue '%1'" [*Error Messages*].	147
SQLE_EVENT_OBJECT_ALREADY_EXISTS	See "Event object named '%1' already exists" [*Error Messages*].	-1265
SQLE_EVENT_PARAMETER_NOT_FOUND	See "Event parameter '%1' not found" [*Error Messages*].	-1267
SQLE_EXPRESSION_ERROR	See "Invalid expression near '%1'" [*Error Messages*].	-156
SQLE_FEATURE_NOT_ENABLED	See "The method you attempted to invoke was not enabled for your application" [*Error Messages*].	-1092
SQLE_FILE_BAD_DB	See "Unable to start specified database: '%1' is not a valid database file" [*Error Messages*].	-1006
SQLE_FILE_IN_USE	See "Specified database file already in use" [*Error Messages*].	-816

Member name	Description	Value
SQLE_FILE_NOT_DB	See "Unable to start specified database: '%1' is not a database" [*Error Messages*].	-1004
SQLE_FILE_VOLUME_NOT_FOUND	See "Specified file system volume not found for database '%1'" [*Error Messages*].	-1112
SQLE_FILE_WRONG_VERSION	See "Unable to start specified database: '%1' was created by a different version of the software" [*Error Messages*].	-1005
SQLE_FOREIGN_KEY_NAME_NOT_FOUND	See "Foreign key name '%1' not found" [*Error Messages*].	-145
SQLE_IDENTIFIER_TOO_LONG	See "Identifier '%1' too long" [*Error Messages*].	-250
SQLE_INCORRECT_VOLUME_ID	See "Incorrect volume ID for '%1'" [*Error Messages*].	-975
SQLE_INDEX_NAME_NOT_UNIQUE	See "Index name '%1' not unique" [*Error Messages*].	-111
SQLE_INDEX_NOT_FOUND	See "Cannot find index named '%1'" [*Error Messages*].	-183
SQLE_INDEX_NOT_UNIQUE	See "Index '%1' for table '%2' would not be unique" [*Error Messages*].	-196
SQLE_INTERRUPTED	See "Statement interrupted by user" [*Error Messages*].	-299
SQLE_INVALID_CONSTRAINT_REF	See "Invalid reference to or operation on constraint '%1'" [*Error Messages*].	-937
SQLE_INVALID_DESCRIPTOR_INDEX	See "Invalid descriptor index" [*Error Messages*].	-640
SQLE_INVALID_DESCRIPTOR_NAME	See "Invalid SQL descriptor name" [*Error Messages*].	-642
SQLE_INVALID_DISTINCT_AGGREGATE	See "Grouped query contains more than one distinct aggregate function" [*Error Messages*].	-863
SQLE_INVALID_EVENT_OBJECT_NAME	See "Event object name '%1' is not valid" [*Error Messages*].	-1264

Member name	Description	Val-ue
SQLE_INVALID_FOREIGN_KEY	See “No primary key value for foreign key '%1' in table '%2'” [*Error Messages*].	-194
SQLE_INVALID_FOR-EIGN_KEY_DEF	See “Column '%1' in foreign key has a different def-inition than primary key” [*Error Messages*].	-113
SQLE_INVALID_GROUP_SELECT	See “Function or column reference to '%1' must also appear in a GROUP BY” [*Error Messages*].	-149
SQLE_INVALID_INDEX_TYPE	See “Index type specification of '%1' is invalid” [*Error Messages*].	-650
SQLE_INVALID_LOGON	See “Invalid user ID or password” [*Error Messag-es*].	-103
SQLE_INVALID_OPTION	See “Invalid option '%1' -- no PUBLIC setting ex-ists” [*Error Messages*].	-200
SQLE_INVALID_OPTION_SETTING	See “Invalid setting for option '%1'” [*Error Mes-sages*].	-201
SQLE_INVALID_OPTION_VALUE	See “'%1' is an invalid value for '%2'” [*Error Mes-sages*].	-105 3
SQLE_INVALID_ORDER	See “Invalid ORDER BY specification” [*Error Messages*].	-152
SQLE_INVALID_PARAMETER	See “Invalid parameter” [*Error Messages*].	-735
SQLE_INVALID_PARSE_PARAME-TER	See “Parse error: %1” [*Error Messages*].	-95
SQLE_INVALID_SQL_IDENTIFIER	See “Invalid SQL identifier” [*Error Messages*].	-760
SQLE_INVALID_STATEMENT	See “Invalid statement” [*Error Messages*].	-130
SQLE_INVALID_UNION	See “Select lists in UNION, INTERSECT, or EX-CEPT do not match in length” [*Error Messages*].	-153
SQLE_KEYLESS_ENCRYPTION	See “Unable to perform requested operation since this database uses keyless encryption” [*Error Mes-sages*].	-110 9
SQLE_LOCKED	See “User '%1' has the row in '%2' locked” [*Error Messages*].	-210

Member name	Description	Value
SQLE_MAX_ROW_SIZE_EXCEEDED	See "Maximum row size for table '%1' would be exceeded" [*Error Messages*].	-1132
SQLE_MEMORY_ERROR	See "Memory error -- transaction rolled back" [*Error Messages*].	-309
SQLE_METHOD_CANNOT_BE_CALLED	See "Method '%1' cannot be called at this time" [*Error Messages*].	-669
SQLE_MIRROR_FILE_MISMATCH	See "The mirror '%1' does not match database '%2'" [*Error Messages*].	-1138
SQLE_MIRROR_FILE_REQUIRES_CHECKSUMS	See "Mirror file requires higher checksum_level" [*Error Messages*].	144
SQLE_MOBILINK_COMMUNICATIONS_ERROR	See "MobiLink communication error; code: %1, parameter: %2, system code %3" [*Error Messages*].	-1305
SQLE_NAME_NOT_UNIQUE	See "Item '%1' already exists" [*Error Messages*].	-110
SQLE_NO_COLUMN_NAME	See "Derived table '%1' has no name for column %2" [*Error Messages*].	-163
SQLE_NO_CURRENT_ROW	See "No current row of cursor" [*Error Messages*].	-197
SQLE_NO_INDICATOR	See "No indicator variable provided for NULL result" [*Error Messages*].	-181
SQLE_NO_MATCHING_SELECT_ITEM	See "The select list for the derived table '%1' has no expression to match '%2'" [*Error Messages*].	-812
SQLE_NO_PRIMARY_KEY	See "Table '%1' has no primary key" [*Error Messages*].	-118
SQLE_NOERROR	SQLE_NOERROR(0) - This code indicates that there was no error or warning.	0
SQLE_NON_UPDATEABLE_COLUMN	See "Cannot update an expression" [*Error Messages*].	-190
SQLE_NON_UPDATEABLE_CURSOR	See "FOR UPDATE has been incorrectly specified for a READ ONLY cursor" [*Error Messages*].	-813
SQLE_NOT_IMPLEMENTED	See "Feature '%1' not implemented" [*Error Messages*].	-134

Member name	Description	Val-ue
SQLE_NOT_SUPPORTED_IN_UL-TRALITE	See "Feature not available with UltraLite" [*Error Messages*].	-749
SQLE_NOTFOUND	See "Row not found" [*Error Messages*].	100
SQLE_ONLY_ONE_TABLE	See "INSERT/DELETE on cursor can modify only one table" [*Error Messages*].	-199
SQLE_OVERFLOW_ERROR	See "Value %1 out of range for destination" [*Error Messages*].	-158
SQLE_PAGE_SIZE_INVALID	See "Invalid database page size" [*Error Messages*].	-644
SQLE_PARTIAL_DOWN-LOAD_NOT_FOUND	See "No partial download was found" [*Error Mes-sages*].	-110 3
SQLE_PASS-THROUGH_SQL_SCRIPT_FAILED_E	See "Passthrough SQL script failed" [*Error Mes-sages*].	-123 8
SQLE_PASS-THROUGH_SQL_SCRIPT_FAILED_W	See "Passthrough SQL script failed" [*Error Mes-sages*].	141
SQLE_PERMISSION_DENIED	See "Permission denied: %1" [*Error Messages*].	-121
SQLE_PRI-MARY_KEY_NOT_UNIQUE	See "Primary key for table '%1' is not unique : Pri-mary key value ('%2')" [*Error Messages*].	-193
SQLE_PRIMARY_KEY_TWICE	See "Table cannot have two primary keys" [*Error Messages*].	-126
SQLE_PRIMARY_KEY_VALUE_REF	See "Primary key for row in table '%1' is referenced by foreign key '%2' in table '%3'" [*Error Messag-es*].	-198
SQLE_PUBLICATION_NOT_FOUND	See "Publication '%1' not found" [*Error Messag-es*].	-280
SQLE_PUBLICATION_PREDI-CATE_IGNORED	See "Publication predicates were not evaluated" [*Error Messages*].	138
SQLE_RESOURCE_GOVERNOR_EX-CEEDED	See "Resource governor for '%1' exceeded" [*Error Messages*].	-685
SQLE_ROW_DELETED_TO_MAIN-TAIN_REFERENTIAL_INTEGRITY	See "Row was dropped from table %1 to maintain referential integrity" [*Error Messages*].	137

Member name	Description	Val-ue
SQLE_ROW_DROPPED_DURING_SCHEMA_UPGRADE	See “A row was dropped because it could not be converted to the new schema format” [*Error Messages*].	130
SQLE_ROW_EXCEEDS_PAGE_SIZE	See “A row cannot be stored because it exceeds the database page size” [*Error Messages*].	-1117
SQLE_ROW_UPDATED_SINCE_READ	See “Row has changed since last read -- operation canceled” [*Error Messages*].	-208
SQLE_SCHEMA_UPGRADE_NOT_ALLOWED	See “A schema upgrade is not currently allowed” [*Error Messages*].	-953
SQLE_SERVER_SYNCHRONIZATION_ERROR	See “Synchronization failed due to an error on the server: %1” [*Error Messages*].	-857
SQLE_START_STOP_DATABASE_DENIED	See “Request to start/stop database denied” [*Error Messages*].	-75
SQLE_STATEMENT_ERROR	See “SQL statement error” [*Error Messages*].	-132
SQLE_STRING_RIGHT_TRUNCATION	See “Right truncation of string data” [*Error Messages*].	-638
SQLE_SUBQUERY_RESULT_NOT_UNIQUE	See “Subquery cannot return more than one row” [*Error Messages*].	-186
SQLE_SUBQUERY_SELECT_LIST	See “Subquery allowed only one select list item” [*Error Messages*].	-151
SQLE_SYNC_INFO_INVALID	See “Information for synchronization is incomplete or invalid, check '%1'” [*Error Messages*].	-956
SQLE_SYNC_INFO_REQUIRED	See “Information for synchronization was not provided” [*Error Messages*].	-1111
SQLE_SYNC_NOT_REENTRANT	See “Synchronization process was unable to re-enter synchronization” [*Error Messages*].	-1110
SQLE_SYNC_PROFILE_INVALID	See “Synchronization profile '%1' has invalid parameter '%2'” [*Error Messages*].	-1224
SQLE_SYNC_PROFILE_NOT_FOUND	See “Synchronization profile '%1' not found” [*Error Messages*].	-1217

Member name	Description	Value
SQLE_SYNC_STATUS_UNKNOWN	See "The status of the last synchronization upload is unknown" [*Error Messages*].	-952
SQLE_SYNTAX_ERROR	See "Syntax error near '%1' %2" [*Error Messages*].	-131
SQLE_TABLE_ALREADY_INCLUDED	See "Table '%1' is already included" [*Error Messages*].	-822
SQLE_TABLE_HAS_PUBLICATIONS	See "Table '%1' has publications" [*Error Messages*].	-281
SQLE_TABLE_IN_USE	See "Table in use" [*Error Messages*].	-214
SQLE_TABLE_NOT_FOUND	See "Table '%1' not found" [*Error Messages*].	-141
SQLE_TOO_MANY_BLOB_REFS	See "Too many references to a BLOB" [*Error Messages*].	-1107
SQLE_TOO_MANY_CONNECTIONS	See "Database server connection limit exceeded" [*Error Messages*].	-102
SQLE_TOO_MANY_CURSORS	See "Too many open cursors" [*Error Messages*].	-1230
SQLE_TOO_MANY_PUBLICATIONS	See "Too many publications specified for operation" [*Error Messages*].	-1106
SQLE_TOO_MANY_TEMP_TABLES	See "Too many temporary tables in connection" [*Error Messages*].	-817
SQLE_TOO_MANY_USERS	See "Too many users in database" [*Error Messages*].	-1104
SQLE_TRUNCATED	See "Value truncated" [*Error Messages*].	101
SQLE_ULTRALITE_DATABASE_NOT_FOUND	See "The database '%1' was not found" [*Error Messages*].	-954
SQLE_ULTRALITE_OBJ_CLOSED	See "Invalid operation on a closed object" [*Error Messages*].	-908
SQLE_UNABLE_TO_CONNECT	See "Database cannot be started -- %1" [*Error Messages*].	-105
SQLE_UNABLE_TO_START_DATABASE	See "Unable to start specified database: %1" [*Error Messages*].	-82

Member name	Description	Value
SQLE_UNABLE_TO_START_DATABASE_VER_NEWER	See "Unable to start specified database: Server must be upgraded to start database %1" [*Error Messages*].	-934
SQLE_UNKNOWN_FUNC	See "Unknown function '%1'" [*Error Messages*].	-148
SQLE_UNKNOWN_OPTION	See "'%1' is an unknown option" [*Error Messages*].	120
SQLE_UNKNOWN_USERID	See "User ID '%1' does not exist" [*Error Messages*].	-140
SQLE_UNRECOGNIZED_OPTION	See "The option '%1' is not recognized" [*Error Messages*].	-1002
SQLE_UPLOAD_FAILED_AT_SERVER	See "Synchronization server failed to commit the upload" [*Error Messages*].	-794
SQLE_VALUE_IS_NULL	See "Cannot return NULL result as requested data type" [*Error Messages*].	-1050
SQLE_VARIABLE_INVALID	See "Invalid host variable" [*Error Messages*].	-155
SQLE_WRONG_NUM_OF_INSERT_COLS	See "Wrong number of values for INSERT" [*Error Messages*].	-207
SQLE_WRONG_PARAMETER_COUNT	See "Wrong number of parameters to function '%1'" [*Error Messages*].	-154

ULSqlPassthroughProgressListener interface

UL Ext.: The listener interface for receiving SQL pass through script progress events.

Syntax

Visual Basic
Public Interface **ULSqlPassthroughProgressListener**

C#
public interface **ULSqlPassthroughProgressListener**

See also

- "ULSqlPassthroughProgressListener members" on page 457

ULSqlPassthroughProgressListener members

Public methods

Member name	Description
"ScriptProgressed method" on page 457	Invoked when SQL pass through scripts are run, to inform the application of progress. This method should return true to cancel further script execution, or return false to continue.

See also

- "ULSqlPassthroughProgressListener interface" on page 457

ScriptProgressed method

Invoked when SQL pass through scripts are run, to inform the application of progress. This method should return true to cancel further script execution, or return false to continue.

Syntax

Visual Basic
```
Public Function ScriptProgressed( _
  ByVal data As ULSqlProgressData _
) As Boolean
```

C#
```
public bool ScriptProgressed(
  ULSqlProgressData data
);
```

Parameters

- **data** A ULSqlProgressData object containing the information about the current script.

Return value

This method should return true to cancel script processing, or return false to continue.

Remarks

No UltraLite.NET API methods should be invoked during a ScriptProgressed call.

See also

- "ULSqlPassthroughProgressListener interface" on page 457
- "ULSqlPassthroughProgressListener members" on page 457
- "ULSqlProgressData class" on page 459

ULSqlProgressData class

UL Ext.: Returns SQL passthrough script progress monitoring data.

Syntax

Visual Basic
Public Class **ULSqlProgressData**

C#
public class **ULSqlProgressData**

See also

- "ULSqlProgressData members" on page 459
- "ULSqlPassthroughProgressListener interface" on page 457

ULSqlProgressData members

Public properties

Member name	Description
"CurrentScript property" on page 459	The index of the scripts executed so far.
"ScriptCount property" on page 460	Returns the number of scripts being executed.
"State property" on page 460	Returns the current progress state.

See also

- "ULSqlProgressData class" on page 459
- "ULSqlPassthroughProgressListener interface" on page 457

CurrentScript property

The index of the scripts executed so far.

Syntax

Visual Basic
Public Readonly Property **CurrentScript** As Long

C#
public long **CurrentScript** { get;}

Property value

The current index of the scripts being executed.

See also

- “ULSqlProgressData class” on page 459
- “ULSqlProgressData members” on page 459

ScriptCount property

Returns the number of scripts being executed.

Syntax

Visual Basic
Public Readonly Property **ScriptCount** As Long

C#
public long **ScriptCount** { get;}

Property value

The number of scripts being executed.

See also

- “ULSqlProgressData class” on page 459
- “ULSqlProgressData members” on page 459

State property

Returns the current progress state.

Syntax

Visual Basic
Public Readonly Property **State** As ULSqlProgressState

C#
public ULSqlProgressState **State** { get;}

Property value

One of the ULSqlProgressState values specifying the current SQL pass through callback state.

See also

- “ULSqlProgressData class” on page 459
- “ULSqlProgressData members” on page 459
- “ULSyncProgressState enumeration” on page 512

ULSqlProgressState enumeration

UL Ext.: Enumerates all the states that can occur while executing SQL pass through scripts.

Syntax

Visual Basic
Public Enum **ULSqlProgressState**

C#
public enum **ULSqlProgressState**

Members

Member name	Description	Value
STATE_DONE	Scripts have successfully completed.	2
STATE_ERROR	Scripts have completed, but an error occurred.	3
STATE_RUN-NING_SCRIPT	Currently running a SQL pass through script.	1
STATE_STARTING	No scripts have been executed yet.	0

See also

- "ULSqlProgressData class" on page 459

ULStreamErrorCode enumeration

UL Ext.: Enumerates the error codes that may be reported by streams during synchronization.

Syntax

Visual Basic
Public Enum **ULStreamErrorCode**

C#
public enum **ULStreamErrorCode**

Members

Member name	Description	Value
ACT-SYNC_NO_PORT	ActiveSync synchronization can only be initiated by ActiveSync itself, either by placing the device in its cradle or by selecting "Synchronize" from the ActiveSync Manager. To initiate a synchronization from an application, use the TCP/IP socket synchronization stream.	75
ACTSYNC_NOT_IN-STALLED	The ActiveSync provider has not been installed. Run mlasinst to install it (see documentation for details).	76
AUTHENTICA-TION_FAILED	The client failed to authenticate itself to MobiLink.	249
CONNECT_TIME-OUT	The connection attempt timed out. Either the server is not running on the indicated host and port or the timeout value needs to increased to allow more time to connect.	241
COULD_NOT_OPEN _FILE	The specified file could not be opened.	264
COULD_NOT_OPEN _FILE_FOR_WRITE	The specified file could not be opened for write. Make sure that this is the correct file and that no other application is using it.	230
CREATE_RAN-DOM_OBJECT	The secure network layer could not create a random-number-generating object. Free up system resources, reconnect and retry the operation.	17

Member name	Description	Value
DEQUEUING_CON-NECTION	The MobiLink server encountered an error while attempting to get a queued connection (synchronization) request. Free up system resources. If the problem persists, restart the MobiLink server.	19
DUN_DIAL_FAILED	Automatic dialup failed to establish connection to the specified dial up network.	204
DUN_NOT_SUPPOR-TED	An attempt to dialup has failed due to insufficient system support. On PocketPC you must use cellcore.dll and on Windows you must use wininet.dll from IE 4.0 or later. Dialup is not supported on other platforms.	203
E2EE_DECOD-ING_PRI-VATE_KEY_FILE	The file was found and its contents were read, but there was an error decoding the file. Please contact technical support and provide the error code.	257
E2EE_INIT_ECC	An error occurred when attempting to initialize ECC. Please make sure the ECC option is installed.	262
E2EE_INVA-LID_TYPE	An invalid e2ee_type was specified. Please specify a valid value.	261
E2EE_MISMATCH-ED_KEYS	The client and server are unable to communicate because the e2ee_public_key used for end-to-end encryption at the remote does not match the e2ee_private_key at the server.	253
E2EE_MISSING_PRI-VATE_KEY	Another end-to-end encryption option was specified, but not the e2ee_private_key option. Either specify all end-to-end encryption options or remove them all. Required end-to-end encryption options include: e2ee_type, e2ee_private_key, e2ee_private_key_password.	260
E2EE_MISSING_PRI-VATE_KEY_PASS-WORD	The e2ee_private_key file cannot be read without an e2ee_private_key_password. Please provide the e2ee_private_key_password.	259
E2EE_NO_PRI-VATE_KEY_IN_FILE	The given filename does not contain a private key.	256
E2EE_PUBLIC_KEY	An error occurred while trying to read the end-to-end encryption public key.	263

Member name	Description	Value
E2EE_READ-ING_PRIVATE_KEY	An error occurred reading the e2ee_private_key file. Please contact technical support and provide the error code.	255
E2EE_READ-ING_PRI-VATE_KEY_FILE	The given file could not be read. Please contact technical support and provide the error code.	258
E2EE_UNEXPEC-TED_PRI-VATE_KEY_TYPE	The private key type found in the e2ee_private_key file does not match the type specified in the e2ee_type.	254
E2EE_UNEXPEC-TED_PUB-LIC_KEY_ENC_TYPE	The client sent an e2ee_type value that is different from the e2ee_type specified at the server. Please make sure these are the same.	252
E2EE_UN-KNOWN_PUB-LIC_KEY_ENC_TYPE	The client sent an e2ee_type value that is not recognized by the server. Please make sure the server version is equal to or greater than the version of the remote.	251
END_READ	Unable to finish a sequence of reads from the network. See also: READ	11
END_WRITE	Unable to finish a sequence of writes to the network. See also: WRITE	10
GENERATE_RAN-DOM	The secure network layer requires a random number but was unable to generate one. Free up system resources, reconnect and retry the operation.	14
HTTP_AUTHENTI-CATION_FAILED	The supplied userid and password were rejected. Check that they were entered correctly. If so, contact your systems administrator to ensure you have proper access.	211
HTTP_AUTHENTI-CATION_REQUIRED	An HTTP server or gateway requested HTTP authentication. Please supply a userid and password using the HTTP synchronization parameters http_userid and http_password.	209
HTTP_BAD_STA-TUS_CODE	Examine the status line to determine the cause of the failure.	86

Member name	Description	Value
HTTP_BUFFER_SIZE_OUT_OF_RANGE	Fix the HTTP buffer size. A valid buffer size is positive and not overly large for the host platform.	79
HTTP_CHUNK_LEN_BAD_CHARACTER	Try using a fixed length HTTP body.	85
HTTP_CHUNK_LEN_ENCODED_MISSING	Try using a fixed length HTTP body.	84
HTTP_CLIENT_ID_NOT_SET	The client id was not passed into the HTTP client code. Contact technical support for a fix.	78
HTTP_CONTENT_TYPE_NOT_SPECIFIED	An unknown content type was specified. Refer to the documentation and change the content type to one of the supported types.	77
HTTP_CRLF_ENCODED_MISSING	The proxy you are using may not be compatible with MobiLink. Please check your configuration.	81
HTTP_CRLF_MISSING	The proxy you are using may not be compatible with MobiLink. Please check your configuration.	82
HTTP_EXPECTED_POST	The proxy you are using may not be compatible with MobiLink. Please check your configuration.	89
HTTP_EXTRA_DATA_END_READ	Extra data has been introduced into the HTTP body. This may have been added by a proxy agent. Try eliminating the proxy.	80
HTTP_HEADER_PARSE_ERROR	An error occurred while trying to parse an HTTP header. The header may be malformed.	216
HTTP_INTERNAL_HEADER_STATE	There was a problem decoding the HTTP header. This is an internal error that should never occur. Please contact technical support.	236
HTTP_INTERNAL_REQUEST_TYPE	There was a problem determining the HTTP request type. This is an internal error that should never occur. Please contact technical support.	237
HTTP_INVALID_CHARACTER	An unexpected character was read in an HTTP header. The header may be malformed or the other side may not be sending HTTP at all.	219

Member name	Description	Value
HTTP_INVA-LID_SESSION_KEY	An unknown session key type was specified. Refer to the documentation and change the session key type to one of the supported types.	244
HTTP_MAL-FORMED_SES-SION_COOKIE	The HTTP cookie used to manage the synchronization session is corrupt. Determine where the cookie is being corrupted. The most likely cause is a client error, or perhaps an HTTP intermediary misbehaving,	235
HTTP_NO_CONTD_CONNECTION	The server timed out while waiting for the next HTTP request from the remote site. Determine why this request failed to reach the server or try a persistent connection.	83
HTTP_NO_PASS-WORD	A userid was supplied for HTTP authentication but no password. Both are required for authentication.	214
HTTP_NO_USERID	A password was supplied for HTTP authentication but no userid. Both are required for authentication.	213
HTTP_PROXY_AU-THENTICA-TION_FAILED	The supplied userid and password were rejected by the proxy server. Check that they were entered correctly. If so, contact your systems administrator to ensure you have proper access.	212
HTTP_PROXY_AU-THENTICATION_RE-QUIRED	An HTTP proxy requested HTTP authentication. Please supply a userid and password using the HTTP synchronization parameters http_proxy_userid and http_proxy_password.	210
HTTP_SERV-ER_AUTH_FAILED	The Authentication-Info header sent from the server contained an incorrect value, causing authentication to fail. Make sure that you are connecting to a legitimate HTTP server.	217
HTTP_UN-ABLE_TO_PARSE_COOKIE	Determine where the set cookie header is being corrupted.	88
HTTP_UN-KNOWN_TRANS-FER_ENCODING	Determine how the unknown transfer encoding is getting generated.	87

Member name	Description	Value
HTTP_UNSUPPORTED_AUTH_ALGORITHM	The HTTP Digest authentication algorithm requested by the server is unsupported. Only "MD5" and "MD5-sess" are supported.	215
HTTP_VERSION	The requested HTTP version is unsupported. Consult the documentation and specify a supported HTTP version. At the time of publication the supported HTTP versions are 1.0 and 1.1.	54
INCONSISTENT_FIPS	Use of the -fips switch on the MobiLink server command line requires that all secure streams be FIPS-compliant. If a secure stream is not configured with the fips option, it will automatically be FIPS-compliant (for example, fips=y). Either remove the fips option from the secure stream, or enable it with fips=y.	242
INIT_RANDOM	The secure network layer could not initialize its random number generator. Free up system resources, reconnect and retry the operation.	15
INTERNAL	An internal error has occurred in the network layer. Please contact technical support.	220
INTERNAL_API	An internal error has occurred in the network layer. Please contact technical support.	238
INTERNAL_PROTOCOL_NOT_LOADED	A synchronization protocol could not be loaded. If you are using UltraLite, make sure you have called the proper ULEnable method.	227
INTERRUPTED	The current operation was interrupted by the caller.	218
INVALID_COMPRESSION_TYPE	The specified compression type was not recognized.	232
INVALID_LOCAL_PATH	The local path for the downloaded file is invalid. Consult the documentation for details.	243
INVALID_NETWORK_LIBRARY	The given network interface DLL or shared object could not be loaded, possibly because it is invalid or corrupt.	245
INVALID_SYNC_PROTOCOL	The specified protocol is not a valid synchronization protocol.	226

Member name	Description	Value
LIBRARY_EN-TRY_POINT_NOT_FOUND	The indicated library entry point could not be found.	225
LOAD_LI-BRARY_FAILURE	The indicated library could not be found in the path. If you are trying to use TLS encryption for syn-chronization, make sure you have acquired the proper license.	224
LOAD_NET-WORK_LIBRARY	The network interface library could not be found and/or loaded. Please check the following: 1) The sockets layer is properly installed. The cor-rect network interface library (or DLL or shared object) must be present and accessible. 2) There are enough system resources available. Free up system resources if they are running low.	73
MEMORY_ALLOCA-TION	The network layer was unable to allocate enough bytes of storage. Free up system memory and retry the operation. The technique used to free up system memory depends on the operating system and how it is configured. The simplest technique is to reduce the number of active processes. Consult your oper-ating system documentation for details.	6
MISSING_PARAME-TER	The specified parameter was expected but not sup-plied.	229
NETWORK_LI-BRARY_VER-SION_MISMATCH	A network interface DLL or shared object could not be loaded because it is the wrong version.	248
NO_ECC_FIPS	There was a problem performing the given com-pression operation. Please contact technical sup-port.	239
NONE	This code indicates there was either no network er-ror, or an unknown network error occurred.	0
NOT_IMPLEMEN-TED	An unimplemented internal feature was requested. Please contact technical support.	12

Member name	Description	Value
PARAMETER	Network parameters are of the form "name=value;[name2=value2[;...]]". This code indicates an invalid parameter value. Consult the documentation for the corresponding parameter name, and correct the parameter value.	1
PARAMETER_NOT_BOOLEAN	Network parameters are of the form "name=value;[name2=value2[;...]]". The parameter value is not a boolean value. Locate the offending parameter specification and change the value of the parameter to either 0 (for off or false) or 1 (for on or true).	4
PARAMETER_NOT_HEX	Network parameters are of the form "name=value;[name2=value2[;...]]". The parameter value is not a hexadecimal (base 16) value. Locate the offending parameter specification and change the value of the parameter to a hexadecimal value.	5
PARAMETER_NOT_UINT32	Network parameters are of the form "name=value;[name2=value2[;...]]". The parameter value is not an unsigned integer. Locate the offending parameter specification and change the value of the parameter to an unsigned integer.	2
PARAMETER_NOT_UINT32_RANGE	Network parameters are of the form "name=value;[name2=value2[;...]]". The parameter value is not an unsigned integer value or range. Locate the offending parameter specification and change the value of the parameter to an unsigned integer or an unsigned range. An unsigned range has the form: NNN-NNN.	3
PARSE	Network parameters are of the form "name=value;[name2=value2[;...]]". Optionally, the entire list of parameters may be enclosed in parentheses. The given string does not follow this convention. Inspect the string, fix any formatting problems, and retry the operation.	7
PROTOCOL_ERROR	An unexpected value or token was read.	231

Member name	Description	Value
READ	Unable to read the given number of bytes from the network layer. Note that reads may occur as part of any larger network operation. For example, some network layers have sub-layers that perform several reads and writes as part of a basic operation in the upper layer. The cause of a read error is usually one of the following: 1) The network had a problem that caused the read to fail. Reconnect and retry the operation. 2) The connection timed out. Reconnect and retry the operation. 3) The other side of the connection cleanly terminated the connection. Consult the client and/or server logs for errors that indicate why the connection has been dropped. Consult the output-log errors and fix the cause, then retry the operation. 4) The process at the other side of the connection was aborted. Consult the client and/or server message logs for errors that indicate why the process was aborted. If the process was shut down by other than normal means, there may not be any errors in its message log. Reconnect and retry the operation. 5) The system is low on resources, and cannot perform the read. Free up system resources, reconnect and retry the operation. If subsequent retry attempts fail, consult your network administrator.	8
READ_TIMEOUT	Unable to read the given number of bytes from the network layer in the given time. Check that the network is functioning correctly, and that the sending application is still running.	201
SACI_ERROR_CLIENT	An error has been reported by the SACI implementation.	246
SACI_ERROR_SERVER	The SACI network library is reporting an error. Refer to the provider of the SACI network library to resolve the problem.	247
SACI_IMPLEMENTATION_MISMATCH	The SACI implementation could not be loaded because it had an incompatible implementation ID.	250

Member name	Description	Value
SECURE_ADD_CERTIFICATE	The secure network layer was unable to add a certificate to a certificate chain. Free up system resources and retry the operation.	39
SECURE_ADD_TRUSTED_CERTIFICATE	The secure network layer was unable to add a trusted certificate to a certificate chain. The most likely cause is a shortage of system resources. Free up system resources and retry the operation.	48
SECURE_CERTIFICATE_COMMON_NAME	The given common name is not in the certificate chain. Check the following: 1) The common name was properly entered. 2) The correct certificate file was specified. 3) The common name is in the certificate chain. You can verify this with the viewcert utility.	52
SECURE_CERTIFICATE_COMPANY_NAME	The given organization name is not in the certificate chain. Check the following: 1) The organization name was properly entered. 2) The correct certificate file was specified. 3) The organization name is in the certificate chain. You can verify this with the viewcert utility.	21
SECURE_CERTIFICATE_COMPANY_UNIT	The given organization unit is not in the certificate chain. Check the following: 1) The in company name was properly entered. 2) The correct certificate file was specified. 3) The company name is in the certificate chain. You can verify this with the viewcert utility.	51
SECURE_CERTIFICATE_COUNT	The given file does not contain a certificate. Check the following: 1) The certificate file name was properly specified. 2) The certificate file contains one or more certificates. 3) The certificate file contains the correct certificate(s).	42

Member name	Description	Value
SECURE_CERTIFI-CATE_EXPIRED	A certificate in the certificate chain has expired. Obtain a new certificate with a later expiry date and retry the operation.	50
SECURE_CERTIFI-CATE_EXPI-RY_DATE	A certificate's expiry date could not be read. Check the following: 1) The password was entered correctly. 2) The certificate file contains one or more certifi-cates. 3) The certificate file contains the correct certifi-cate(s). 4) The certificate file is undamaged.	37
SECURE_CERTIFI-CATE_FILE_NOT_FO UND	The certificate file could not be opened. Check the following: 1) The certificate file name was properly specified. 2) The certificate file exists. 3) The certificate file contains one or more certifi-cates. 4) The certificate file contains the correct certifi-cate(s). 5) The program attempting to open the certificate file has sufficient privileges to read the file. This only applies to operating systems having user and/ or file permissions.	33
SECURE_CERTIFI-CATE_NOT_TRUS-TED	The server's certificate was not signed by a trusted authority. Check the following: 1) The certificate file name was properly specified. 2) The certificate file contains one or more certifi-cates. 3) The certificate file contains the correct certifi-cate(s). 4) The client's list of trusted root certificates in-cludes the server's root certificate.	24
SECURE_CERTIFI-CATE_ROOT	The root certificate in the chain is invalid. At the time of publication, this error was defined but not used.	20

Member name	Description	Value
SECURE_CREATE_CERTIFICATE	The secure network layer was unable to allocate storage for a certificate. Free up system resources and retry the operation.	43
SECURE_CREATE_PRIVATE_KEY_OBJECT	The secure network layer was unable to create a private key object, prior to loading the private key. The most likely cause is a shortage of system resources. Free up system resources and retry the operation.	49
SECURE_DUPLICATE_CONTEXT	The secure network layer was unable to duplicate a security context. Free up system resources and retry the operation.	25
SECURE_EXPORT_CERTIFICATE	The secure network layer was unable to copy a certificate. Free up system resources and retry the operation.	38
SECURE_HANDSHAKE	The secure handshake failed. Check the following: 1) On the client, the correct host machine and port number were specified. 2) On the server, the correct port number was specified. 3) The correct trusted certificate was specified on the client and the correct identity file was specified on the server.	53
SECURE_IMPORT_CERT_FROM_SYSTEM_STORE	Failed to import a certificate from the system certificate store.	222
SECURE_IMPORT_CERTIFICATE	The secure network layer was unable to import a certificate. Check the following: 1) The certificate file name was properly specified. 2) The certificate file exists. 3) The certificate file contains one or more certificates. 4) The certificate file contains the correct certificate(s).	44

Member name	Description	Value
SE-CURE_NO_CERTS_IN_SYS_STORE	No certificates were found in the system's certificate store.	223
SECURE_NO_SERVER_CERTIFICATE	No server certificate was provided. A server certificate is required for secure communications. The file provided must contain the full chain of certificates for the server and its private key.	205
SECURE_NO_SERVER_CERTIFICATE_PASSWORD	No server certificate password was provided. This password is required to decrypt the server's encrypted private key.	206
SECURE_NO_TRUSTED_ROOTS	No trusted root certificates were provided. At least one trusted root certificate is required for secure communications.	207
SECURE_OPEN_SYSTEM_CERT_STORE	An attempt to open a system certificate store failed.	221
SECURE_READ_CERTIFICATE	The certificate file could not be read. Check the following: 1) The password was entered correctly. 2) The certificate file contains one or more certificates. 3) The certificate file contains the correct certificate(s). 4) The certificate file is undamaged.	34
SECURE_READ_PRIVATE_KEY	The private key could not be read from the certificate file. Check the following: 1) The password was entered correctly. 2) The certificate file contains one or more certificates. 3) The certificate file contains the correct certificate(s). 4) The certificate file is undamaged.	35

Member name	Description	Value
SECURE_REDUNDANT_SERVER_CERTIFICATE_PASSWORD	A password was specified when the server's private key wasn't encrypted by any password.	208
SECURE_SET_IO	The secure network layer was unable to attach to the network layer. Free up system resources and retry the operation.	26
SECURE_SET_PRIVATE_KEY	The private key could not be used. Check the following: 1) The password was entered correctly. 2) The certificate file contains one or more certificates. 3) The certificate file contains the correct certificate(s). 4) The certificate file is undamaged.	36
SECURE_TRUSTED_CERTIFICATE_FILE_NOT_FOUND	The certificate file could not be found. Check the following: 1) The certificate file name was properly specified. 2) The certificate file exists. 3) The certificate file contains one or more certificates. 4) The certificate file contains the correct certificate(s). 5) The program attempting to open the certificate file has sufficient privileges to see the file. This only applies to operating systems having user and/or file permissions.	40

Member name	Description	Value
SECURE_TRUS-TED_CERTIFI-CATE_READ	The secure network layer was unable to read the trusted certificate file. Check the following: 1) The certificate file name was properly specified. 2) The certificate file exists. 3) The certificate file contains one or more certifi-cates. 4) The certificate file contains the correct certifi-cate(s). 5) The program attempting to open the certificate file has sufficient privileges to see the file. This only applies to operating systems having user and/or file permissions.	41
SEED_RANDOM	The secure network layer could not seed its random number generator. Free up system resources, re-connect and retry the operation.	16
SERVER_ERROR	The server reported an error. Contact the MobiLink administrator to learn more.	228
SHUTTING_DOWN	The MobiLink server encountered an error in the network layer during shutdown. It is possible that some network operations pending at the time of shutdown were affected.	18

Member name	Description	Value
SOCKET_BIND	The network layer was unable to bind a socket to the given port. Check the following. 1) (Server only) Verify that the port isn't already in use. If the port is in use, either shut down the application listening on that port, or specify a different port. 2) (Server only) Verify that there are no firewall restrictions on the use of the port. 3) (Client only) If the client_port option was used, verify that the given port isn't already in use. If only one client port was specified, consider using a range (for example, NNN-NNN). If a range was specified, consider making it a wider range, or a different range. 4) (Client only) If the client_port option was used, verify that there are no firewall restrictions on the use of the port.	60
SOCKET_CLEANUP	The network layer was unable to clean up the socket layer. This error should only occur after all connections are finished, so no current connections should be affected.	61
SOCKET_CLOSE	The network layer was unable to close a socket. The network session may or may not have terminated prematurely, due to pending writes that were not flushed. Check the following: 1) The other side of the network connection had any errors. 2) The other side of the connection is running normally. 3) The machine is still connected to the network, and the network is responsive.	62

Member name	Description	Value
SOCKET_CONNECT	The network layer was unable to connect a socket. Check the following: 1) The machine is connected to the network. 2) The socket layer is properly initialized. 3) The correct host machine and port were specified. 4) The host server is running normally and listening on the correct port. 5) The host machine is listening for the proper socket type (TCP/IP vs. UDP). 6) If the client_port option was used, verify that there are no firewall restrictions on the use of the port. 7) If the device has a limit on the number of open sockets, verify that the limit has not been reached. 8) There are enough system resources available. Free up system resources if they are running low.	63
SOCKET_CREATE_TCPIP	The network layer was unable to create a TCP/IP socket. Check the following: 1) The machine is connected to the network. 2) The socket layer is properly initialized. 5) If the device has a limit on the number of open sockets, verify that the limit has not been reached. 6) There are enough system resources available. Free up system resources if they are running low.	58

Member name	Description	Value
SOCKET_CRE-ATE_UDP	The network layer was unable to create a UDP socket. Check the following: 1) The machine is connected to the network. 2) The socket layer is properly initialized. 3) If the client_port option was used, verify that the given port isn't already in use. If only one client port was specified, consider using a range (for example, NNN-NNN). If a range was specified, consider making it a wider range, or a different range. 4) If the client_port option was used, verify that there are no firewall restrictions on the use of the port. 5) If the device has a limit on the number of open sockets, verify that the limit has not been reached. 6) There are enough system resources available. Free up system resources if they are running low.	59
SOCK-ET_GET_HOST_BY_ADDR	The network layer was unable to get the name of a host using its IP address. At the time of publication, this error was defined but not used.	72
SOCK-ET_GET_NAME	The network layer was unable to determine a sock-et's local name. In a TCP/IP connection, each end of the connection has a socket exclusively attached to a port. A socket's local name includes this port number, which is assigned by the network at con-nection time. Check the following: 1) The machine is still connected to the network, and the network is responsive. 2) The other side of the connection is running nor-mally. 3) There are enough system resources available. Free up system resources if they are running low.	64

Member name	Description	Value
SOCKET_GET_OPTION	The network layer was unable to get a socket option. This error may be the first indication that a connection has been lost. Check the following: 1) The machine is still connected to the network, and the network is responsive. 2) The other side of the connection is running normally. 3) There are enough system resources available. Free up system resources if they are running low.	65
SOCKET_HOST_NAME_NOT_FOUND	The given host name could not be found. Check the following: 1) The host name was correctly specified. 2) The host is accessible. Many systems include a "ping" utility that can be used to verify access to a named host. 3) The Domain Name Server (DNS), or its equivalent, is available. If the DNS is not available, try specifying the host's IP number (for example, NNN.NNN.NNN.NNN) instead of the host name. 4) The HOSTS file contains an entry that maps the host name to an IP number.	57
SOCKET_LISTEN	The server is unable to listen on a socket. The backlog refers to the maximum number of queued connection requests that may be pending at any given time. Check the following: 1) The machine is still connected to the network, and the network is responsive. 2) There are no firewall or other restrictions preventing a socket listener from running on the current machine. 3) The backlog setting is within the limit, if any, on the machine. 4) There are enough system resources available. Free up system resources if they are running low.	67
SOCKET_LIVENESS_OUT_OF_RANGE	An invalid liveness timeout value was specified. The liveness timeout value must be an integer between zero and 65535.	200

Member name	Description	Value
SOCKET_LOCAL-HOST_NAME_NOT_FOUND	The network layer was unable to determine the IP address of "localhost". Check the following: 1) The Domain Name Server (DNS), or its equivalent, is available. If the DNS is not available, try explicitly specifying the localhost IP number (usually 127.0.0.1) instead. 2) The HOSTS file contains an entry that maps the "localhost" name to an IP number. 3) There are enough system resources available. Free up system resources if they are running low.	71
SOCK-ET_PORT_OUT_OF_RANGE	An invalid port number was specified. The port number must be an integer between zero and 65535.	74
SOCKET_SELECT	The network layer encountered an error attempting to wait for a socket to be ready for reading or writing. Check the following: 1) The machine is connected to the network, and the network is responsive. 2) The other side of the connection is running normally. 3) There are enough system resources available. Free up system resources if they are running low.	69
SOCKET_SET_OP-TION	The network layer was unable to set a socket option. This error may be the first indication that a connection has been lost. Check the following: 1) The machine is still connected to the network, and the network is responsive. 2) The other side of the connection is running normally. 3) There are enough system resources available. Free up system resources if they are running low.	66

Member name	Description	Value
SOCKET_SHUT-DOWN	The network layer was unable to shut down a socket. Check the following: 1) The machine is connected to the network, and the network is responsive. 2) The other side of the connection is running normally. 3) There are enough system resources available. Free up system resources if they are running low.	68
SOCKET_STARTUP	The network layer was unable to initialize the socket layer. Check the following: 1) The sockets layer is properly installed. The correct network interface library must be present and accessible. 2) The machine is connected to the network, and the network is responsive. 3) There are enough system resources available. Free up system resources if they are running low.	70
UNEXPEC-TED_HTTP_RE-QUEST_TYPE	The given HTTP request type was unexpected at this time. The most likely cause is an HTTP client that is not a MobiLink client.	234
UNRECOG-NIZED_TLS_TYPE	The TLS type is invalid. Consult the documentation for valid types.	240
VAL-UE_OUT_OF_RANGE	The specified value was not in the range of acceptable values for that parameter. Check the documentation for the parameter to learn the acceptable range of values.	233
WOULD_BLOCK	A requested operation would block where blocking is undesirable or unexpected.	13

Member name	Description	Value
WRITE	Unable to write the given number of bytes to the network layer. Note that writes may occur as part of any larger network operation. For example, some network layers have sub-layers that perform several reads and writes as part of a basic operation in the upper layer. The cause of a write error is usually one of the following: 1) The network had a problem that caused the write to fail. Reconnect and retry the operation. 2) The connection timed out. Reconnect and retry the operation. 3) The other side of the connection cleanly terminated the connection. Consult the client and/or server logs for errors that indicate why the connection has been dropped. Consult the output-log errors and fix the cause, then retry the operation. 4) The process at the other side of the connection was aborted. Consult the client and/or server message logs for errors that indicate why the process was aborted. If the process was shut down by other than normal means, there may not be any errors in its message log. Reconnect and retry the operation. 5) The system is low on resources, and cannot perform the write. Free up system resources, reconnect and retry the operation. If subsequent retry attempts fail, consult your network administrator.	9
WRITE_TIMEOUT	Unable to write the given number of bytes to the network layer in the given time. Check that the network is functioning correctly, and that the receiving application is still running.	202

See also

- "StreamErrorCode property" on page 517

ULStreamType enumeration

UL Ext.: Enumerates the types of MobiLink synchronization streams to use for synchronization.

Syntax

Visual Basic
Public Enum **ULStreamType**

C#
public enum **ULStreamType**

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 11 - Introduction*].

Members

Member name	Description	Value
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.	1
HTTPS	Synchronize via HTTPS (HTTP with transport-layer security).	2
TCPIP	Synchronize via TCP/IP.	0
TLS	Synchronize via TCP/IP with transport layer security.	3

When using encrypted synchronization over HTTPS and TLS streams, you must deploy the appropriate DLL file to the device. The following table displays the DLL file to deploy for each type of encryption protocol. DLL files are located in the *UltraLite\CE* directory of your SQL Anywhere installation.

Protocol	**DLL**
ECC	*mlcecc10.dll*
RSA	*mlcrsa10.dll*
FIPS	*mlcrsafips10.dll*

See also

- "Stream property" on page 495

ULSyncParms class

UL Ext.: Represents synchronization parameters that define how to synchronize an UltraLite database. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULSyncParms**

C#
public sealed class **ULSyncParms**

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

- “ULSyncParms members” on page 486
- “ULConnection class” on page 131
- “SyncParms property” on page 143
- “Synchronize() method” on page 172
- “DownloadOnly property” on page 489
- “PingOnly property” on page 492
- “ResumePartialDownload property” on page 493
- “UploadOnly property” on page 496
- “ULSQLCode enumeration” on page 445
- “Synchronize() method” on page 172
- “Stream property” on page 495
- “Version property” on page 498

ULSyncParms members

Public properties

Member name	Description
“AdditionalParms property” on page 488	Specifies additional parameters as a semicolon-separated list of name=value pairs. These are less commonly used parameters.

Member name	Description
“AuthenticationParms property” on page 488	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
“DownloadOnly property” on page 489	Specifies whether to disable or enable uploads when synchronizing.
“KeepPartialDownload property” on page 490	Specifies whether to disable or enable partial downloads when synchronizing.
“NewPassword property” on page 491	Specifies a new MobiLink password for the user specified with UserName.
“Password property” on page 492	The MobiLink password for the user specified by UserName.
“PingOnly property” on page 492	Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.
“Publications property” on page 493	Specifies the publications to be synchronized.
“ResumePartialDownload property” on page 493	Specifies whether to resume or discard a previous partial download.
“SendColumnNames property” on page 494	Specifies whether the client should send column names to the MobiLink server during synchronization.
“SendDownloadAck property” on page 495	Specifies whether the client should send a download acknowledgement to the MobiLink server during synchronization. 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).
“Stream property” on page 495	Specifies the MobiLink synchronization stream to use for synchronization.
“StreamParms property” on page 496	Specifies the parameters to configure the synchronization stream.
“UploadOnly property” on page 496	Specifies whether to disable or enable downloads when synchronizing.
“UserName property” on page 497	The user name that uniquely identifies the MobiLink client to the MobiLink server.
“Version property” on page 498	Specifies which synchronization script to use.

Public methods

Member name	Description
"CopyFrom method" on page 498	Copies the properties of the specified ULSyncParms object to this ULSyncParms object.

See also

- "ULSyncParms class" on page 486
- "ULConnection class" on page 131
- "SyncParms property" on page 143
- "Synchronize() method" on page 172
- "DownloadOnly property" on page 489
- "PingOnly property" on page 492
- "ResumePartialDownload property" on page 493
- "UploadOnly property" on page 496
- "ULSQLCode enumeration" on page 445
- "Synchronize() method" on page 172
- "Stream property" on page 495
- "Version property" on page 498

AdditionalParms property

Specifies a string of keyword=value pairs. This list of pairs usually contains synchronization parameters that are infrequently used.

Syntax

Visual Basic
Public Property **AdditionalParms** As String

C#
public string **AdditionalParms** { get; set; }

Property value

A semicolon-separated list of keyword=value additional parameters.

See also

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Syntax

Visual Basic
Public Property **AuthenticationParms** As String()

C#
public string[] **AuthenticationParms** { get; set; }

Property value

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.

Remarks

Only the first 255 strings are used and each string should be no longer than 128 characters (longer strings are truncated when sent to the MobiLink server).

See also

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486

DownloadOnly property

Specifies whether to disable or enable uploads when synchronizing.

Syntax

Visual Basic
Public Property **DownloadOnly** As Boolean

C#
public bool **DownloadOnly** { get; set; }

Property value

True to disable uploads when synchronizing, false to enable uploads. The default is false.

Remarks

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

- “ULSyncParms class” on page 486
- “ULSyncParms members” on page 486
- “UploadOnly property” on page 496
- “DownloadOnly property” on page 489
- “PingOnly property” on page 492
- “ResumePartialDownload property” on page 493
- “UploadOnly property” on page 496
- “ULSQLCode enumeration” on page 445
- “Synchronize() method” on page 172

KeepPartialDownload property

Specifies whether to disable or enable partial downloads when synchronizing.

Syntax

Visual Basic
Public Property **KeepPartialDownload** As Boolean

C#
public bool **KeepPartialDownload** { get; set; }

Property value

True to enable partial downloads when synchronizing, false to disable partial downloads. The default is false.

Remarks

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 fields KeepPartialDownload and DisableConcurrency 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

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486
- "PartialDownloadRetained property" on page 516
- "ResumePartialDownload property" on page 493
- "RollbackPartialDownload method" on page 170
- "ResumePartialDownload property" on page 493
- "UserName property" on page 497
- "AdditionalParms property" on page 488
- "RollbackPartialDownload method" on page 170

NewPassword property

Specifies a new MobiLink password for the user specified with UserName.

Syntax

Visual Basic
Public Property **NewPassword** As String

C#
public string **NewPassword** { get; set; }

Property value

A string specifying a new MobiLink password. The default is a null reference (Nothing in Visual Basic), meaning the password is not changed.

Remarks

A new password takes effect after the next synchronization.

See also

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486
- "UserName property" on page 497

Password property

The MobiLink password for the user specified by UserName.

Syntax

Visual Basic
Public Property **Password** As String

C#
public string **Password** { get; set; }

Property value

A string specifying the MobiLink password. The default is a null reference (Nothing in Visual Basic), meaning no password is specified.

Remarks

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

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486
- "NewPassword property" on page 491
- "UserName property" on page 497

PingOnly property

Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.

Syntax

Visual Basic
Public Property **PingOnly** As Boolean

C#
public bool **PingOnly** { get; set; }

Property value

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.

Remarks

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

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486
- "DownloadOnly property" on page 489
- "PingOnly property" on page 492
- "ResumePartialDownload property" on page 493
- "UploadOnly property" on page 496
- "ULSQLCode enumeration" on page 445
- "Synchronize() method" on page 172

Publications property

Specifies the publications to be synchronized.

Syntax

Visual Basic
Public Property **Publications** As String

C#
public string **Publications** { get; set; }

Property value

A string containing a list of publication names, separated by comma (,); or the special value ULPublicationSchema.SYNC_ALL_PUBS, or the special value ULPublicationSchema.SYNC_ALL_DB. The default is ULPublicationSchema.SYNC_ALL_DB. For more information, see "ULPublicationSchema class" on page 395.

See also

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486
- "SYNC_ALL_PUBS field" on page 396
- "SYNC_ALL_DB field" on page 396
- "ULPublicationSchema class" on page 395

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Syntax

Visual Basic
Public Property **ResumePartialDownload** As Boolean

C#
public bool **ResumePartialDownload** { get; set; }

Property value

True to resume a previous partial download, false to discard a previous partial download. The default is false.

Remarks

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().

For more information about partial downloads, see "KeepPartialDownload property" on page 490.

See also

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486
- "DownloadOnly property" on page 489
- "PingOnly property" on page 492
- "ResumePartialDownload property" on page 493
- "UploadOnly property" on page 496
- "ULSQLCode enumeration" on page 445
- "Synchronize() method" on page 172
- "PartialDownloadRetained property" on page 516

SendColumnNames property

Specifies whether the client should send column names to the MobiLink server during synchronization.

Syntax

Visual Basic
Public Property **SendColumnNames** As Boolean

C#
public bool **SendColumnNames** { get; set; }

Property value

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.

Remarks

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.

See also

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486

SendDownloadAck property

Specifies whether the client should send a download acknowledgement to the MobiLink server during synchronization. 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).

Syntax

Visual Basic
Public Property **SendDownloadAck** As Boolean

C#
public bool **SendDownloadAck** { get; set; }

Property value

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.

Remarks

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.

See also

- “ULSyncParms class” on page 486
- “ULSyncParms members” on page 486

Stream property

Specifies the MobiLink synchronization stream to use for synchronization.

Syntax

Visual Basic
Public Property **Stream** As ULStreamType

C#
public ULStreamType **Stream** { get; set; }

Property value

One of the ULStreamType values specifying the type of synchronization stream to use. The default is ULStreamType.TCPIP.

Remarks

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

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486
- "ULStreamType enumeration" on page 484
- "StreamParms property" on page 496
- "ULStreamType enumeration" on page 484

StreamParms property

Specifies the parameters to configure the synchronization stream.

Syntax

Visual Basic
Public Property **StreamParms** As String

C#
public string **StreamParms** { get; set; }

Property value

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).

Remarks

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

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486
- "Stream property" on page 495
- "ULStreamType enumeration" on page 484

UploadOnly property

Specifies whether to disable or enable downloads when synchronizing.

Syntax

Visual Basic
Public Property **UploadOnly** As Boolean

C#
public bool **UploadOnly** { get; set; }

Property value

True to disable downloads, false to enable downloads. The default is false.

Remarks

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

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486
- "DownloadOnly property" on page 489
- "PingOnly property" on page 492
- "ResumePartialDownload property" on page 493
- "UploadOnly property" on page 496
- "ULSQLCode enumeration" on page 445
- "Synchronize() method" on page 172

UserName property

The user name that uniquely identifies the MobiLink client to the MobiLink server.

Syntax

Visual Basic
Public Property **UserName** As String

C#
public string **UserName** { get; set; }

Property value

A string specifying the user name. This parameter has no default value, and must be explicitly set.

Remarks

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

- “ULSyncParms class” on page 486
- “ULSyncParms members” on page 486
- “Password property” on page 492

Version property

Specifies which synchronization script to use.

Syntax

Visual Basic

```
Public Property Version As String
```

C#

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

Property value

A string specifying the version of the synchronization script to use. This parameter has no default value, and must be explicitly set.

Remarks

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.

See also

- “ULSyncParms class” on page 486
- “ULSyncParms members” on page 486

CopyFrom method

Copies the properties of the specified ULSyncParms object to this ULSyncParms object.

Syntax

Visual Basic

```
Public Sub CopyFrom( _
  ByVal src As ULSyncParms _
)
```

C#

```
public void CopyFrom(
  ULSyncParms src
);
```

Parameters

- **src** The object to copy from.

See also

- "ULSyncParms class" on page 486
- "ULSyncParms members" on page 486
- "ULSyncParms class" on page 486

ULSyncProgressData class

UL Ext.: Returns synchronization progress monitoring data.

Syntax

Visual Basic
Public Class **ULSyncProgressData**

C#
public class **ULSyncProgressData**

See also

- “ULSyncProgressData members” on page 500
- “ULSyncProgressListener interface” on page 510

ULSyncProgressData members

Public fields

Member name	Description
“FLAG_IS_BLOCKING field” on page 501	A flag indicating that the synchronization is blocked awaiting a response from the MobiLink server. This field is constant and read-only.

Public properties

Member name	Description
“Flags property” on page 502	Returns the current synchronization flags indicating additional information relating to the current state.
“ReceivedBytes property” on page 502	Returns the number of bytes received so far. This information is updated for all states.
“ReceivedDeletes property” on page 503	Returns the number of deleted rows received so far.
“ReceivedInserts property” on page 503	Returns the number of inserted rows received so far.
“ReceivedUpdates property” on page 503	Returns the number of updated rows received so far.
“SentBytes property” on page 504	Returns the number of bytes sent so far. This information is updated for all states.

Member name	Description
"SentDeletes property" on page 504	Returns the number of deleted rows sent so far.
"SentInserts property" on page 505	Returns the number of inserted rows sent so far.
"SentUpdates property" on page 505	Returns the number of updated rows sent so far.
"State property" on page 506	Returns the current synchronization state.
"SyncTableCount property" on page 506	Returns the number of tables being synchronized.
"SyncTableIndex property" on page 507	Returns the index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).
"TableID property" on page 507	Returns the database index of the table currently being synchronized.
"TableName property" on page 508	Returns the name of the current table being uploaded or downloaded.

Public methods

Member name	Description
"SetValues method" on page 508	Load current sync values into this object

See also

- "ULSyncProgressData class" on page 500
- "ULSyncProgressListener interface" on page 510

FLAG_IS_BLOCKING field

A flag indicating that the synchronization is blocked awaiting a response from the MobiLink server. This field is constant and read-only.

Syntax

Visual Basic
Public Shared **FLAG_IS_BLOCKING** As Integer

C#
public const int **FLAG_IS_BLOCKING**;

See also

- “ULSyncProgressData class” on page 500
- “ULSyncProgressData members” on page 500

Flags property

Returns the current synchronization flags indicating additional information relating to the current state.

Syntax

Visual Basic
Public Readonly Property **Flags** As Integer

C#
public int **Flags** { get;}

Property value

An integer containing a combination of flags or'ed together.

See also

- “ULSyncProgressData class” on page 500
- “ULSyncProgressData members” on page 500
- “FLAG_IS_BLOCKING field” on page 501

ReceivedBytes property

Returns the number of bytes received so far. This information is updated for all states.

Syntax

Visual Basic
Public Readonly Property **ReceivedBytes** As Long

C#
public long **ReceivedBytes** { get;}

Property value

The number of bytes received so far.

See also

- “ULSyncProgressData class” on page 500
- “ULSyncProgressData members” on page 500
- “ULSyncProgressState enumeration” on page 512
- “ULSyncProgressState enumeration” on page 512

ReceivedDeletes property

Returns the number of deleted rows received so far.

Syntax

Visual Basic
Public Readonly Property **ReceivedDeletes** As Integer

C#
public int **ReceivedDeletes** { get;}

Property value

The number of deleted rows received so far.

See also

- "ULSyncProgressData class" on page 500
- "ULSyncProgressData members" on page 500
- "ULSyncProgressState enumeration" on page 512
- "ULSyncProgressState enumeration" on page 512

ReceivedInserts property

Returns the number of inserted rows received so far.

Syntax

Visual Basic
Public Readonly Property **ReceivedInserts** As Integer

C#
public int **ReceivedInserts** { get;}

Property value

The number of inserted rows received so far.

See also

- "ULSyncProgressData class" on page 500
- "ULSyncProgressData members" on page 500
- "ULSyncProgressState enumeration" on page 512
- "ULSyncProgressState enumeration" on page 512

ReceivedUpdates property

Returns the number of updated rows received so far.

Syntax

Visual Basic
Public Readonly Property **ReceivedUpdates** As Integer

C#
public int **ReceivedUpdates** { get;}

Property value

The number of updated rows received so far.

See also

- "ULSyncProgressData class" on page 500
- "ULSyncProgressData members" on page 500
- "ULSyncProgressState enumeration" on page 512
- "ULSyncProgressState enumeration" on page 512

SentBytes property

Returns the number of bytes sent so far. This information is updated for all states.

Syntax

Visual Basic
Public Readonly Property **SentBytes** As Long

C#
public long **SentBytes** { get;}

Property value

The number of bytes sent so far.

See also

- "ULSyncProgressData class" on page 500
- "ULSyncProgressData members" on page 500
- "ULSyncProgressState enumeration" on page 512
- "ULSyncProgressState enumeration" on page 512

SentDeletes property

Returns the number of deleted rows sent so far.

Syntax

Visual Basic
Public Readonly Property **SentDeletes** As Integer

C#
public int **SentDeletes** { get;}

Property value

The number of deleted rows sent so far.

See also

- "ULSyncProgressData class" on page 500
- "ULSyncProgressData members" on page 500
- "ULSyncProgressState enumeration" on page 512
- "ULSyncProgressState enumeration" on page 512

SentInserts property

Returns the number of inserted rows sent so far.

Syntax

Visual Basic
Public Readonly Property **SentInserts** As Integer

C#
public int **SentInserts** { get;}

Property value

The number of inserted rows sent so far.

See also

- "ULSyncProgressData class" on page 500
- "ULSyncProgressData members" on page 500
- "ULSyncProgressState enumeration" on page 512
- "ULSyncProgressState enumeration" on page 512

SentUpdates property

Returns the number of updated rows sent so far.

Syntax

Visual Basic
Public Readonly Property **SentUpdates** As Integer

C#
public int **SentUpdates** { get;}

Property value

The number of updated rows sent so far.

See also

- “ULSyncProgressData class” on page 500
- “ULSyncProgressData members” on page 500
- “ULSyncProgressState enumeration” on page 512
- “ULSyncProgressState enumeration” on page 512

State property

Returns the current synchronization state.

Syntax

Visual Basic
Public Readonly Property **State** As ULSyncProgressState

C#
public ULSyncProgressState **State** { get;}

Property value

One of the ULSyncProgressState values specifying the current synchronization state.

See also

- “ULSyncProgressData class” on page 500
- “ULSyncProgressData members” on page 500
- “ULSyncProgressState enumeration” on page 512

SyncTableCount property

Returns the number of tables being synchronized.

Syntax

Visual Basic
Public Readonly Property **SyncTableCount** As Integer

C#
public int **SyncTableCount** { get;}

Property value

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

- “ULSyncProgressData class” on page 500
- “ULSyncProgressData members” on page 500
- “ULSyncProgressState enumeration” on page 512
- “ULSyncProgressState enumeration” on page 512

SyncTableIndex property

Returns the index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).

Syntax

Visual Basic
Public Readonly Property **SyncTableIndex** As Integer

C#
public int **SyncTableIndex** { get;}

Property value

The index of the table currently being synchronized, in the range from 1 to SyncTableCount

See also

- "ULSyncProgressData class" on page 500
- "ULSyncProgressData members" on page 500
- "ULSyncProgressState enumeration" on page 512
- "ULSyncProgressState enumeration" on page 512

TableID property

Returns the database index of the table currently being synchronized.

Syntax

Visual Basic
Public Readonly Property **TableID** As Integer

C#
public int **TableID** { get;}

Property value

The database index, in the range from 1 to ULDatabaseSchema.TableCount

See also

- "ULSyncProgressData class" on page 500
- "ULSyncProgressData members" on page 500
- "ULSyncProgressState enumeration" on page 512
- "ULSyncProgressState enumeration" on page 512
- "TableCount property" on page 254

TableName property

Returns the name of the table currently being uploaded or downloaded.

Syntax

Visual Basic
Public Property **TableName** As String

C#
public string **TableName** { get; set; }

Property value

The name of the table being uploaded or downloaded. Null, if not applicable.

SetValues method

Load current sync values into this object

Syntax

Visual Basic
Public Sub **SetValues(** _
  ByVal *sync_state* As ULSyncProgressState, _
  ByVal *table_id* As Integer, _
  ByVal *table_count* As Integer, _
  ByVal *table_index* As Integer, _
  ByVal *sent_bytes* As Long, _
  ByVal *sent_inserts* As Integer, _
  ByVal *sent_updates* As Integer, _
  ByVal *sent_deletes* As Integer, _
  ByVal *recv_bytes* As Long, _
  ByVal *recv_inserts* As Integer, _
  ByVal *recv_updates* As Integer, _
  ByVal *recv_deletes* As Integer, _
  ByVal *flag_s* As Integer _
)

C#
public void **SetValues(**
  ULSyncProgressState *sync_state*,
  int *table_id*,
  int *table_count*,
  int *table_index*,
  long *sent_bytes*,
  int *sent_inserts*,
  int *sent_updates*,
  int *sent_deletes*,
  long *recv_bytes*,
  int *recv_inserts*,
  int *recv_updates*,
  int *recv_deletes*,

```
  int flag_s
);
```

See also

- “ULSyncProgressData class” on page 500
- “ULSyncProgressData members” on page 500

ULSyncProgressListener interface

UL Ext.: The listener interface for receiving synchronization progress events.

Syntax

Visual Basic
Public Interface **ULSyncProgressListener**

C#
public interface **ULSyncProgressListener**

See also

- "ULSyncProgressListener members" on page 510
- "Synchronize(ULSyncProgressListener) method" on page 173

ULSyncProgressListener members

Public methods

Member name	Description
"SyncProgressed method" on page 510	Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue.

See also

- "ULSyncProgressListener interface" on page 510
- "Synchronize(ULSyncProgressListener) method" on page 173

SyncProgressed method

Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue.

Syntax

Visual Basic
Public Function **SyncProgressed(** _
  ByVal *data* As ULSyncProgressData _
) As Boolean

C#
public bool **SyncProgressed(**
  ULSyncProgressData *data*
);

Parameters

- **data** A ULSyncProgressData object containing the latest synchronization progress data.

Return value

This method should return true to cancel synchronization or return false to continue.

Remarks

No UltraLite.NET API methods should be invoked during a SyncProgressed call.

See also

- "ULSyncProgressListener interface" on page 510
- "ULSyncProgressListener members" on page 510
- "ULSyncProgressData class" on page 500

ULSyncProgressState enumeration

UL Ext.: Enumerates all the states that can occur while synchronizing.

Syntax

Visual Basic
Public Enum **ULSyncProgressState**

C#
public enum **ULSyncProgressState**

Members

Member name	Description	Value
STATE_canceled	Synchronization has been canceled.	15
STATE_COMMIT-TING_DOWNLOAD	The download is being committed. The final count of rows received is included with this event.	9
STATE_CONNECT-ING	The synchronization stream has been built, but is not yet opened.	1
STATE_DISCON-NECTING	The synchronization stream is about to be closed.	11
STATE_DONE	Synchronization has successfully completed.	12
STATE_ERROR	Synchronization has completed, but an error occur-red.	13
STATE_FINISH-ING_UPLOAD	The upload is completing. The final count of rows sent is included with this event.	5
STATE_RECEIV-ING_DATA	Data for the current table is being received. UL-SyncProgressData.ReceivedBytes, ULSyncProg-ressData.ReceivedInserts, ULSyncProgressDa-ta.ReceivedUpdates, and ULSyncProgressDa-ta.ReceivedDeletes have been updated.	8
STATE_RECEIV-ING_TABLE	A table is being received. Progress can be moni-tored using ULSyncProgressData.SyncTableIndex and ULSyncProgressData.SyncTableCount.	7
STATE_RECEIV-ING_UPLOAD_ACK	An acknowledgement that the upload is complete is being received.	6

Member name	Description	Value
STATE_ROLL-ING_BACK_DOWN-LOAD	Synchronization is rolling back the download because an error was encountered during the download. The error will be reported with a subsequent STATE_ERROR progress report.	14
STATE_SEND-ING_DATA	Data for the current table is being sent. ULSync-ProgressData.SentBytes, ULSyncProgressData.SentInserts, ULSyncProgressData.SentUpdates, and ULSyncProgressData.SentDeletes have been updated.	4
STATE_SEND-ING_DOWN-LOAD_ACK	An acknowledgement that the download is complete is being sent.	10
STATE_SEND-ING_HEADER	The synchronization stream has been opened and the header is about to be sent.	2
STATE_SEND-ING_TABLE	A table is being sent. Progress can be monitored using ULSyncProgressData.SyncTableIndex and ULSyncProgressData.SyncTableCount.	3
STATE_STARTING	No synchronization actions have been taken yet.	0

See also

- "ULSyncProgressData class" on page 500

ULSyncResult class

UL Ext.: Represents the status of the last synchronization.

Syntax

Visual Basic
Public Class **ULSyncResult**

C#
public class **ULSyncResult**

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

- "ULSyncResult members" on page 514
- "SyncResult property" on page 144
- "Synchronize() method" on page 172

ULSyncResult members

Public properties

Member name	Description
"AuthStatus property" on page 515	Returns the authorization status code for the last synchronization attempt.
"AuthValue property" on page 515	Returns the return value from custom user authentication synchronization scripts.
"IgnoredRows property" on page 516	Checks whether any uploaded rows were ignored during the last synchronization.
"PartialDownloadRetained property" on page 516	Checks whether a partial download was retained during the last synchronization.
"StreamErrorCode property" on page 517	Returns the error reported by the stream itself.
"StreamErrorParameters property" on page 517	Returns a comma-separated list of stream error parameters.
"StreamErrorSystem property" on page 518	Returns the stream error system-specific code.

Member name	Description
"Timestamp property" on page 518	Returns the timestamp of the last synchronization.
"UploadOK property" on page 519	Checks whether the last upload synchronization was successful.

See also

- "ULSyncResult class" on page 514
- "SyncResult property" on page 144
- "Synchronize() method" on page 172

AuthStatus property

Returns the authorization status code for the last synchronization attempt.

Syntax

Visual Basic
Public Readonly Property **AuthStatus** As ULAuthStatusCode

C#
public ULAuthStatusCode **AuthStatus** { get;}

Property value

One of the ULAuthStatusCode values denoting the authorization status for the last synchronization attempt.

See also

- "ULSyncResult class" on page 514
- "ULSyncResult members" on page 514
- "ULAuthStatusCode enumeration" on page 56

AuthValue property

Returns the return value from custom user authentication synchronization scripts.

Syntax

Visual Basic
Public Readonly Property **AuthValue** As Long

C#
public long **AuthValue** { get;}

Property value

A long integer returned from custom user authentication synchronization scripts.

See also

- "ULSyncResult class" on page 514
- "ULSyncResult members" on page 514

IgnoredRows property

Checks whether any uploaded rows were ignored during the last synchronization.

Syntax

Visual Basic
Public Readonly Property **IgnoredRows** As Boolean

C#
public bool **IgnoredRows** { get;}

Property value

True if any uploaded rows were ignored during the last synchronization, false if no rows were ignored.

See also

- "ULSyncResult class" on page 514
- "ULSyncResult members" on page 514
- "DownloadOnly property" on page 489

PartialDownloadRetained property

Checks whether a partial download was retained during the last synchronization.

Syntax

Visual Basic
Public Readonly Property **PartialDownloadRetained** As Boolean

C#
public bool **PartialDownloadRetained** { get;}

Property value

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

- “ULSyncResult class” on page 514
- “ULSyncResult members” on page 514
- “KeepPartialDownload property” on page 490

StreamErrorCode property

Returns the error reported by the stream itself.

Syntax

Visual Basic
Public Readonly Property **StreamErrorCode** As ULStreamErrorCode

C#
public ULStreamErrorCode **StreamErrorCode** { get;}

Property value

One of the ULStreamErrorCode values denoting the error reported by the stream itself, ULStreamErrorCode.NONE if no error occurred.

See also

- “ULSyncResult class” on page 514
- “ULSyncResult members” on page 514
- “ULStreamErrorCode enumeration” on page 462
- “ULStreamErrorCode enumeration” on page 462

StreamErrorParameters property

Returns a comma-separated list of stream error parameters.

Syntax

Visual Basic
Public Readonly Property **StreamErrorParameters** As String

C#
public string **StreamErrorParameters** { get;}

Property value

Contains a comma separated list of error parameters for the stream error code reported in “StreamErrorCode property” on page 517. This will be an empty string for errors with no parameters or when no error has been set.

See also

- “ULSyncResult class” on page 514
- “ULSyncResult members” on page 514
- “ULStreamErrorCode enumeration” on page 462

StreamErrorSystem property

Returns the stream error system-specific code.

Syntax

Visual Basic
Public Readonly Property **StreamErrorSystem** As Integer

C#
public int **StreamErrorSystem** { get;}

Property value

An integer denoting the stream error system-specific code.

See also

- “ULSyncResult class” on page 514
- “ULSyncResult members” on page 514

Timestamp property

Returns the timestamp of the last synchronization.

Syntax

Visual Basic
Public Readonly Property **Timestamp** As Date

C#
public DateTime **Timestamp** { get;}

Property value

A System.DateTime specifying the timestamp of the last synchronization.

See also

- “ULSyncResult class” on page 514
- “ULSyncResult members” on page 514
- DateTime

UploadOK property

Checks whether the last upload synchronization was successful.

Syntax

Visual Basic
Public Readonly Property **UploadOK** As Boolean

C#
public bool **UploadOK** { get;}

Property value

True if the last upload synchronization was successful, false if the last upload synchronization was unsuccessful.

See also

- "ULSyncResult class" on page 514
- "ULSyncResult members" on page 514

ULTable class

UL Ext.: Represents a table in an UltraLite database.

Syntax

Visual Basic
Public Class **ULTable**
  Inherits ULResultSet

C#
public class **ULTable**: ULResultSet

Remarks

There is no constructor for this class. Tables are created using the ULCommand.ExecuteTable() of the ULCommand.

Inherits: ULResultSet

Implements: System.Data.IDataReader, System.Data.IDataRecord, System.IDisposable

See also

- "ULTable members" on page 520
- "ExecuteTable() method" on page 118
- "ULCommand class" on page 86
- "ULResultSet class" on page 398
- IDataReader
- IDataRecord
- IDisposable

ULTable members

Public properties

Member name	Description
"Depth property" on page 266 (inherited from ULDataReader)	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
"FieldCount property" on page 266 (inherited from ULDataReader)	Returns the number of columns in the cursor.
"HasRows property" on page 267 (inherited from ULDataReader)	Checks whether the ULDataReader has one or more rows.

Member name	Description
"IsBOF property" on page 267 (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is before the first row.
"IsClosed property" on page 268 (inherited from ULDataReader)	Checks whether the cursor is currently open.
"IsEOF property" on page 268 (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is after the last row.
"Item properties" on page 268 (inherited from ULDataReader)	Gets the value of a specified column as an instance of Object.
"RecordsAffected property" on page 270 (inherited from ULDataReader)	Returns the number of rows changed, inserted, or deleted by execution of the SQL statement. For SELECT statements or CommandType.TableDirect tables, this value is -1.
"RowCount property" on page 271 (inherited from ULDataReader)	**UL Ext.:** Returns the number of rows in the cursor.
"Schema property" on page 528	Holds the table schema. This property is only valid while its connection is open.
VisibleFieldCount (inherited from DbDataReader)	Gets the number of fields in the DbDataReader that are not hidden.

Public methods

Member name	Description
"AppendBytes method" on page 404 (inherited from ULResultSet)	Appends the specified subset of the specified array of System.Bytes to the new value for the specified ULDbType.LongBinary column.
"AppendChars method" on page 406 (inherited from ULResultSet)	Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType.LongVarchar column.
"Close method" on page 272 (inherited from ULDataReader)	Closes the cursor.
"Delete method" on page 407 (inherited from ULResultSet)	Deletes the current row.

Member name	Description
"DeleteAllRows method" on page 528	Deletes all rows in the table.
Dispose (inherited from DbDataReader)	Releases all resources used by the current instance of the DbDataReader.
"FindBegin method" on page 529	Prepares to perform a new Find on a table.
"FindFirst methods" on page 529	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.
"FindLast methods" on page 531	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 methods" on page 533	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 methods" on page 534	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" on page 272 (inherited from ULDataReader)	Returns the value for the specified column as a System.Boolean.
"GetByte method" on page 273 (inherited from ULDataReader)	Returns the value for the specified column as an unsigned 8-bit value (System.Byte).
"GetBytes methods" on page 273 (inherited from ULDataReader)	**UL Ext.:** Returns the value for the specified column as an array of System.Bytes. Only valid for columns of type ULDbType.Binary, ULDbType.LongBinary, or ULDbType.UniqueIdentifier.
"GetChar method" on page 276 (inherited from ULDataReader)	This method is not supported in UltraLite.NET.
"GetChars method" on page 277 (inherited from ULDataReader)	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 (inherited from DbDataReader)	Returns a DbDataReader object for the requested column ordinal.

Member name	Description
"GetDataTypeName method" on page 278 (inherited from ULDataReader)	Returns the name of the specified column's provider data type.
"GetDateTime method" on page 279 (inherited from ULDataReader)	Returns the value for the specified column as a System.DateTime with millisecond accuracy.
"GetDecimal method" on page 279 (inherited from ULDataReader)	Returns the value for the specified column as a System.Decimal.
"GetDouble method" on page 280 (inherited from ULDataReader)	Returns the value for the specified column as a System.Double.
"GetEnumerator method" on page 281 (inherited from ULDataReader)	Returns an System.Collections.IEnumerator that iterates through the ULDataReader.
"GetFieldType method" on page 281 (inherited from ULDataReader)	Returns the System.Type most appropriate for the specified column.
"GetFloat method" on page 282 (inherited from ULDataReader)	Returns the value for the specified column as a System.Single.
"GetGuid method" on page 282 (inherited from ULDataReader)	Returns the value for the specified column as a UUID (System.Guid).
"GetInt16 method" on page 283 (inherited from ULDataReader)	Returns the value for the specified column as a System.Int16.
"GetInt32 method" on page 284 (inherited from ULDataReader)	Returns the value for the specified column as an Int32.
"GetInt64 method" on page 285 (inherited from ULDataReader)	Returns the value for the specified column as an Int64.
"GetName method" on page 285 (inherited from ULDataReader)	Returns the name of the specified column.
"GetOrdinal method" on page 286 (inherited from ULDataReader)	Returns the column ID of the named column.

Member name	Description
GetProviderSpecificFieldType (inherited from DbDataReader)	Returns the provider-specific field type of the specified column.
GetProviderSpecificValue (inherited from DbDataReader)	Gets the value of the specified column as an instance of Object.
GetProviderSpecificValues (inherited from DbDataReader)	Gets all provider-specific attribute columns in the collection for the current row.
"GetSchemaTable method" on page 288 (inherited from ULDataReader)	Returns a System.Data.DataTable that describes the column metadata of the ULDataReader.
"GetString method" on page 290 (inherited from ULDataReader)	Returns the value for the specified column as a System.String.
"GetTimeSpan method" on page 290 (inherited from ULDataReader)	Returns the value for the specified column as a System.TimeSpan with millisecond accuracy.
"GetUInt16 method" on page 291 (inherited from ULDataReader)	Returns the value for the specified column as a System.UInt16.
"GetUInt32 method" on page 292 (inherited from ULDataReader)	Returns the value for the specified column as a UInt32.
"GetUInt64 method" on page 292 (inherited from ULDataReader)	Returns the value for the specified column as a System.UInt64.
"GetValue method" on page 293 (inherited from ULDataReader)	Returns the value of the specified column in its native format.
"GetValues method" on page 294 (inherited from ULDataReader)	Returns all the column values for the current row.
"Insert method" on page 536	Inserts a new row with the current column values (specified using the set methods). Each insert must be preceded by a call to ULTable.InsertBegin.
"InsertBegin method" on page 536	Prepares to insert a new row into the table by setting all current column values to their default values.

Member name	Description
"IsDBNull method" on page 295 (inherited from ULDataReader)	Checks whether the value from the specified column is NULL.
"LookupBackward methods" on page 537	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" on page 539	Prepares to perform a new lookup on the table. 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.
"LookupForward methods" on page 539	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" on page 295 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to after the last row of the cursor.
"MoveBeforeFirst method" on page 296 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to before the first row of the cursor.
"MoveFirst method" on page 296 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the first row of the cursor.
"MoveLast method" on page 296 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the last row of the cursor.
"MoveNext method" on page 297 (inherited from ULDataReader)	**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" on page 297 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the previous row or before the first row.
"MoveRelative method" on page 298 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor relative to the current row.

Member name	Description
"NextResult method" on page 299 (inherited from ULDataReader)	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
"Read method" on page 299 (inherited from ULDataReader)	Positions the cursor to the next row, or after the last row if the cursor was already on the last row.
"SetBoolean method" on page 408 (inherited from ULResultSet)	Sets the value for the specified column using a System.Boolean.
"SetByte method" on page 409 (inherited from ULResultSet)	Sets the value for the specified column using a System.Byte (unsigned 8-bit integer).
"SetBytes method" on page 409 (inherited from ULResultSet)	Sets the value for the specified column using an array of System.Bytes.
"SetDBNull method" on page 410 (inherited from ULResultSet)	Sets a column to NULL.
"SetDateTime method" on page 411 (inherited from ULResultSet)	Sets the value for the specified column using a System.DateTime.
"SetDecimal method" on page 412 (inherited from ULResultSet)	Sets the value for the specified column using a System.Decimal.
"SetDouble method" on page 413 (inherited from ULResultSet)	Sets the value for the specified column using a System.Double.
"SetFloat method" on page 414 (inherited from ULResultSet)	Sets the value for the specified column using a System.Single.
"SetGuid method" on page 415 (inherited from ULResultSet)	Sets the value for the specified column using a System.Guid.
"SetInt16 method" on page 416 (inherited from ULResultSet)	Sets the value for the specified column using an System.Int16.
"SetInt32 method" on page 417 (inherited from ULResultSet)	Sets the value for the specified column using an System.Int32.

Member name	Description
"SetInt64 method" on page 418 (inherited from ULResultSet)	Sets the value for the specified column using an Int64.
"SetString method" on page 419 (inherited from ULResultSet)	Sets the value for the specified column using a System.String.
"SetTimeSpan meth-od" on page 420 (inherited from ULResultSet)	Sets the value for the specified column using a System.TimeSpan.
"SetToDefault meth-od" on page 420 (inherited from ULResultSet)	Sets the value for the specified column to its default value.
"SetUInt16 meth-od" on page 421 (inherited from ULResultSet)	Sets the value for the specified column using a System.UInt16.
"SetUInt32 meth-od" on page 422 (inherited from ULResultSet)	Sets the value for the specified column using an System.UInt32.
"SetUInt64 meth-od" on page 423 (inherited from ULResultSet)	Sets the value for the specified column using a System.UInt64.
"Truncate meth-od" on page 541	Deletes all rows in the table while temporarily activating a stop syn-chronization delete.
"Update method" on page 424 (inherited from ULResultSet)	Updates the current row with the current column values (specified using the set methods).
"UpdateBegin meth-od" on page 541	Prepares to update the current row in the table.

See also

- "ULTable class" on page 520
- "ExecuteTable() method" on page 118
- "ULCommand class" on page 86
- "ULResultSet class" on page 398
- IDataReader
- IDataRecord
- IDisposable

Schema property

Holds the table schema. This property is only valid while its connection is open.

Syntax

Visual Basic
Public Readonly Property **Schema** As ULTableSchema

C#
public ULTableSchema **Schema** { get;}

Property value

The ULTableSchema object representing the table schema.

Remarks

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "ULTableSchema class" on page 542

DeleteAllRows method

Deletes all rows in the table.

Syntax

Visual Basic
Public Sub **DeleteAllRows()**

C#
public void **DeleteAllRows();**

Remarks

In some applications, it can be useful to delete all rows from a table before downloading a new set of data into the table. Rows can be deleted from the UltraLite database without being deleted from the consolidated database using ULConnection.StopSynchronizationDelete.

See also

- "ULTable class" on page 520
- "ULTable members" on page 520
- "Truncate method" on page 541
- "StopSynchronizationDelete method" on page 172

FindBegin method

Prepares to perform a new Find on a table.

Syntax

Visual Basic
Public Sub **FindBegin()**

C#
public void **FindBegin();**

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "FindFirst() method" on page 529
- "FindFirst(Int16) method" on page 530
- "FindLast() method" on page 531
- "FindLast(Int16) method" on page 532

FindFirst methods

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() method

Moves forward through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.

Syntax

Visual Basic
Public Function **FindFirst()** As Boolean

C#
public bool **FindFirst();**

Return value

True if successful, false otherwise.

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "FindFirst methods" on page 529
- "FindBegin method" on page 529
- "FindNext() method" on page 533
- "FindPrevious() method" on page 535
- "FindFirst(Int16) method" on page 530
- "IsEOF property" on page 268
- "FindBegin method" on page 529

FindFirst(Int16) method

Moves forward through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.

Syntax

Visual Basic

```
Public Function FindFirst( _
  ByVal numColumns As Short _
) As Boolean
```

C#

```
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.

Return value

True if successful, false otherwise.

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "FindFirst methods" on page 529
- "FindBegin method" on page 529
- "FindNext(Int16) method" on page 534
- "FindPrevious(Int16) method" on page 535
- "FindFirst() method" on page 529
- "IsEOF property" on page 268
- "FindBegin method" on page 529

FindLast methods

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.

FindLast() method

Moves backward through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.

Syntax

Visual Basic
Public Function **FindLast()** As Boolean

C#
public bool **FindLast();**

Return value

True if successful, false otherwise.

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "FindLast methods" on page 531
- "FindBegin method" on page 529
- "FindNext() method" on page 533
- "FindPrevious() method" on page 535
- "FindLast(Int16) method" on page 532
- "IsBOF property" on page 267
- "FindBegin method" on page 529

FindLast(Int16) method

Moves backward through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.

Syntax

Visual Basic

Public Function **FindLast(** _
  ByVal *numColumns* As Short _
) As Boolean

C#

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.

Return value

True if successful, false otherwise

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "FindLast methods" on page 531
- "FindBegin method" on page 529
- "FindNext(Int16) method" on page 534
- "FindPrevious(Int16) method" on page 535
- "FindLast() method" on page 531
- "IsBOF property" on page 267
- "FindBegin method" on page 529

FindNext methods

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() 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.

Syntax

Visual Basic
Public Function **FindNext()** As Boolean

C#
public bool **FindNext();**

Return value

True if successful, false otherwise.

Remarks

The cursor is left on the next row if it exactly matches the index value. On failure, the cursor position is after the last row (ULDataReader.IsEOF).

FindNext behavior is undefined if the column values being searched for are modified during a row update.

See also

- "ULTable class" on page 520
- "ULTable members" on page 520
- "FindNext methods" on page 533
- "FindFirst() method" on page 529
- "FindNext(Int16) method" on page 534
- "IsEOF property" on page 268

FindNext(Int16) 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.

Syntax

Visual Basic

```
Public Function FindNext( _
  ByVal numColumns As Short _
) As Boolean
```

C#

```
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.

Return value

True if successful, false otherwise.

Remarks

The cursor is left on the next row if it exactly matches the index value. On failure, the cursor position is after the last row (ULDataReader.IsEOF).

FindNext behavior is undefined if the column values being searched for are modified during a row update.

See also

- "ULTable class" on page 520
- "ULTable members" on page 520
- "FindNext methods" on page 533
- "FindFirst(Int16) method" on page 530
- "FindNext() method" on page 533
- "FindFirst() method" on page 529
- "IsEOF property" on page 268

FindPrevious methods

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() 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.

Syntax

Visual Basic
Public Function **FindPrevious()** As Boolean

C#
public bool **FindPrevious();**

Return value

True if successful, false otherwise.

Remarks

The cursor is left on the previous row if it exactly matches the index value. On failure, the cursor position is before the first row (ULDataReader.IsBOF).

FindPrevious behavior is undefined if the column values being searched for are modified during a row update.

See also

- "ULTable class" on page 520
- "ULTable members" on page 520
- "FindPrevious methods" on page 534
- "FindLast() method" on page 531
- "FindPrevious(Int16) method" on page 535
- "IsBOF property" on page 267

FindPrevious(Int16) 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.

Syntax

Visual Basic
Public Function **FindPrevious(** _
  ByVal *numColumns* As Short _
) As Boolean

C#
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.

Return value

True if successful, false otherwise.

Remarks

The cursor is left on the previous row if it exactly matches the index value. On failure, the cursor position is before the first row (ULDataReader.IsBOF).

FindPrevious behavior is undefined if the column values being searched for are modified during a row update.

See also

- "ULTable class" on page 520
- "ULTable members" on page 520
- "FindPrevious methods" on page 534
- "FindLast() method" on page 531
- "FindLast(Int16) method" on page 532
- "FindPrevious() method" on page 535
- "IsBOF property" on page 267

Insert method

Inserts a new row with the current column values (specified using the set methods).

Each insert must be preceded by a call to ULTable.InsertBegin.

Syntax

Visual Basic
Public Sub **Insert()**

C#
public void **Insert();**

See also

- "ULTable class" on page 520
- "ULTable members" on page 520
- "InsertBegin method" on page 536

InsertBegin method

Prepares to insert a new row into the table by setting all current column values to their default values.

Syntax

Visual Basic
Public Sub **InsertBegin()**

C#
public void **InsertBegin();**

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "Insert method" on page 536
- "Insert method" on page 536

LookupBackward methods

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() method

Moves backward through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.

Syntax

Visual Basic
Public Function **LookupBackward()** As Boolean

C#
public bool **LookupBackward();**

Return value

True if successful, false otherwise.

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "LookupBackward methods" on page 537
- "LookupBegin method" on page 539
- "LookupBackward(Int16) method" on page 538
- "IsBOF property" on page 267

LookupBackward(Int16) method

Moves backward through the table from the beginning, looking for a row that matches or is less than a value or partial set of values in the current index.

Syntax

Visual Basic
Public Function **LookupBackward(** _
  ByVal *numColumns* As Short _
) As Boolean

C#
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.

Return value

True if successful, false otherwise.

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "LookupBackward methods" on page 537
- "LookupBegin method" on page 539
- "LookupBackward(Int16) method" on page 538
- "IsBOF property" on page 267

LookupBegin method

Prepares to perform a new lookup on the table. 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.

Syntax

Visual Basic
Public Sub **LookupBegin()**

C#
public void **LookupBegin();**

See also

- “ULTable class” on page 520
- “ULTable members” on page 520
- “LookupForward() method” on page 539
- “LookupForward(Int16) method” on page 540
- “LookupBackward() method” on page 537
- “LookupBackward(Int16) method” on page 538

LookupForward methods

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() method

Moves forward through the table from the beginning, looking for a row that matches or is greater than a value or full set of values in the current index.

Syntax

Visual Basic
Public Function **LookupForward()** As Boolean

C#
public bool **LookupForward();**

Return value

True if successful, false otherwise.

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "LookupForward methods" on page 539
- "LookupBegin method" on page 539
- "LookupForward(Int16) method" on page 540
- "IsEOF property" on page 268

LookupForward(Int16) method

Moves forward through the table from the beginning, looking for a row that matches or is greater than a value or partial set of values in the current index.

Syntax

Visual Basic

Public Function **LookupForward(** _
  ByVal *numColumns* As Short _
) As Boolean

C#

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.

Return value

True if successful, false otherwise.

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

- "ULTable class" on page 520
- "ULTable members" on page 520
- "LookupForward methods" on page 539
- "LookupBegin method" on page 539
- "LookupForward() method" on page 539
- "IsEOF property" on page 268
- "LookupBegin method" on page 539

Truncate method

Deletes all rows in the table while temporarily activating a stop synchronization delete.

Syntax

Visual Basic
Public Sub **Truncate()**

C#
public void **Truncate();**

See also

- “ULTable class” on page 520
- “ULTable members” on page 520
- “DeleteAllRows method” on page 528

UpdateBegin method

Prepares to update the current row in the table.

Syntax

Visual Basic
Public Sub **UpdateBegin()**

C#
public void **UpdateBegin();**

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 ULResultSet.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

- “ULTable class” on page 520
- “ULTable members” on page 520
- “Update method” on page 424

ULTableSchema class

UL Ext.: Represents the schema of an UltraLite table. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULTableSchema**
  Inherits ULCursorSchema

C#
public sealed class **ULTableSchema**: ULCursorSchema

Remarks

There is no constructor for this class. A ULTableSchema object is attached to a table as its ULTable.Schema.

Inherits: ULCursorSchema

See also

- "ULTableSchema members" on page 542
- "ULTableSchema class" on page 542
- "Schema property" on page 528
- "ULCursorSchema class" on page 220

ULTableSchema members

Public properties

Member name	Description
"ColumnCount property" on page 221 (inherited from ULCursorSchema)	Returns the number of columns in the cursor.
"IndexCount property" on page 544	Returns the number of indexes on the table.
"IsNeverSynchronized property" on page 545	Checks whether the table is marked as never being synchronized.
"IsOpen property" on page 222 (inherited from ULCursorSchema)	Checks whether the cursor schema is currently open.
"Name property" on page 545	Returns the name of the table.
"PrimaryKey property" on page 546	Returns the index schema of the primary key for the table.

Member name	Description
"UploadUnchangedRows property" on page 546	Checks whether the database uploads rows that have not changed.

Public methods

Member name	Description
"GetColumnDefaultValue method" on page 547	Returns the default value of the specified column.
"GetColumnID method" on page 222 (inherited from ULCursorSchema)	Returns the column ID of the named column.
"GetColumnName method" on page 223 (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
"GetColumnPartitionSize method" on page 547	Returns the global autoincrement partition size assigned to the specified column.
"GetColumnPrecision method" on page 224 (inherited from ULCursorSchema)	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnSQLName method" on page 225 (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
"GetColumnScale method" on page 226 (inherited from ULCursorSchema)	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
"GetColumnSize method" on page 226 (inherited from ULCursorSchema)	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
"GetColumnULDbType method" on page 227 (inherited from ULCursorSchema)	Returns the UltraLite.NET data type of the column identified by the specified column ID.
"GetIndex method" on page 548	Returns the index schema of the named index.
"GetIndexName method" on page 549	Returns the name of the index identified by the specified index ID.

Member name	Description
"GetOptimalIndex method" on page 549	The optimal index for searching a table using the specified column.
"GetPublicationPredicate method" on page 550	Returns the publication predicate for this table in the named publication.
"GetSchemaTable method" on page 227 (inherited from ULCursorSchema)	Returns a System.Data.DataTable that describes the column schema of the ULDataReader.
"IsColumnAutoIncrement method" on page 551	Checks whether the specified column's default is set to autoincrement.
"IsColumnCurrentDate method" on page 551	Checks whether the specified column's default is set to the current date (ULDbType.Date).
"IsColumnCurrentTime method" on page 552	Checks whether the specified column's default is set to the current time (ULDbType.Time).
"IsColumnCurrentTimestamp method" on page 552	Checks whether the specified column's default is set to the current timestamp (ULDbType.TimeStamp).
"IsColumnGlobalAutoIncrement method" on page 553	Checks whether the specified column's default is set to global autoincrement.
"IsColumnNewUUID method" on page 554	Checks whether the specified column's default is set to a new UUID (System.Guid).
"IsColumnNullable method" on page 554	Checks whether the specified column is nullable.
"IsInPublication method" on page 555	Checks whether the table is contained in the named publication.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema class" on page 542
- "Schema property" on page 528
- "ULCursorSchema class" on page 220

IndexCount property

Returns the number of indexes on the table.

Syntax

Visual Basic
Public Readonly Property **IndexCount** As Integer

C#
public int **IndexCount** { get;}

Property value

The number of indexes on the table or 0 if the table schema is closed.

Remarks

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.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542

IsNeverSynchronized property

Checks whether the table is marked as never being synchronized.

Syntax

Visual Basic
Public Readonly Property **IsNeverSynchronized** As Boolean

C#
public bool **IsNeverSynchronized** { get;}

Property value

True if the table is marked as never being synchronized, false otherwise.

Remarks

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.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542

Name property

Returns the name of the table.

Syntax

Visual Basic
Public Overrides Readonly Property **Name** As String

C#
public override string **Name** { get;}

Property value

The name of the table as a string.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542

PrimaryKey property

Returns the index schema of the primary key for the table.

Syntax

Visual Basic
Public Readonly Property **PrimaryKey** As ULIndexSchema

C#
public ULIndexSchema **PrimaryKey** { get;}

Property value

A ULIndexSchema object representing the primary key for the table.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "ULIndexSchema class" on page 336

UploadUnchangedRows property

Checks whether the database uploads rows that have not changed.

Syntax

Visual Basic
Public Readonly Property **UploadUnchangedRows** As Boolean

C#
public bool **UploadUnchangedRows** { get;}

Property value

True if the table is marked to always upload all rows during synchronization, false if the table is marked to upload only changed rows.

Remarks

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.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542

GetColumnDefaultValue method

Returns the default value of the specified column.

Syntax

Visual Basic

```
Public Function GetColumnDefaultValue( _
  ByVal columnID As Integer _
) As String
```

C#

```
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.

Return value

The default value of the specified column as a string or a null reference (Nothing in Visual Basic) if the default value is null.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "ColumnCount property" on page 221

GetColumnPartitionSize method

Returns the global autoincrement partition size assigned to the specified column.

Syntax

Visual Basic
```
Public Function GetColumnPartitionSize( _
  ByVal columnID As Integer _
) As UInt64
```

C#
```
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.

Return value

The column's global autoincrement partition size as a System.UInt64.

Remarks

All global autoincrement columns in a given table share the same global autoincrement partition.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "IsColumnGlobalAutoIncrement method" on page 553
- "ColumnCount property" on page 221
- UInt64

GetIndex method

Returns the index schema of the named index.

Syntax

Visual Basic
```
Public Function GetIndex( _
  ByVal name As String _
) As ULIndexSchema
```

C#
```
public ULIndexSchema GetIndex(
  string  name
);
```

Parameters

- **name** The name of the index.

Return value

A ULIndexSchema object representing the named index.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "ULIndexSchema class" on page 336

GetIndexName method

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

Syntax

Visual Basic
Public Function **GetIndexName(** _
  ByVal *indexID* As Integer _
) As String

C#
public string **GetIndexName(**
  int *indexID*
);

Parameters

- **indexID**   The ID of the index. The value must be in the range [1,IndexCount].

Return value

The name of the index as a string.

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

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "IndexCount property" on page 544

GetOptimalIndex method

The optimal index for searching a table using the specified column.

Syntax

Visual Basic
Public Function **GetOptimalIndex(** _

```
  ByVal columnID As Integer _
) As ULIndexSchema
```

C#

```
public ULIndexSchema 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.

Return value

A ULIndexSchema object representing the optimal index for the specified column.

Remarks

The specified column is the first column in the index, but the index may have more than one column.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "ColumnCount property" on page 221
- "ULIndexSchema class" on page 336

GetPublicationPredicate method

Returns the publication predicate for this table in the named publication.

Syntax

Visual Basic

```
Public Function GetPublicationPredicate( _
  ByVal pubName As String _
) As String
```

C#

```
public string  GetPublicationPredicate(
  string  pubName
);
```

Parameters

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

Return value

The publication predicate as a string.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542

IsColumnAutoIncrement method

Checks whether the specified column's default is set to autoincrement.

Syntax

Visual Basic
Public Function **IsColumnAutoIncrement(** _
  ByVal *columnID* As Integer _
) As Boolean

C#
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.

Return value

True if the column is autoincrementing, false if it is not autoincrementing.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "ColumnCount property" on page 221

IsColumnCurrentDate method

Checks whether the specified column's default is set to the current date (ULDbType.Date).

Syntax

Visual Basic
Public Function **IsColumnCurrentDate(** _
  ByVal *columnID* As Integer _
) As Boolean

C#
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.

Return value

True if the column defaults to the current date, false if the column does not default to the current date.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "ColumnCount property" on page 221
- "ULDbType enumeration" on page 301

IsColumnCurrentTime method

Checks whether the specified column's default is set to the current time (ULDbType.Time).

Syntax

Visual Basic

```
Public Function IsColumnCurrentTime( _
  ByVal columnID As Integer _
) As Boolean
```

C#

```
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.

Return value

True if the column defaults to the current time, false if the column does not default to the current time.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "ColumnCount property" on page 221
- "ULDbType enumeration" on page 301

IsColumnCurrentTimestamp method

Checks whether the specified column's default is set to the current timestamp (ULDbType.TimeStamp).

Syntax

Visual Basic
Public Function **IsColumnCurrentTimestamp(** _
  ByVal *columnID* As Integer _
) As Boolean

C#
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.

Return value

True if the column defaults to the current timestamp, false if the column does not default to the current timestamp.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "ColumnCount property" on page 221
- "ULDbType enumeration" on page 301

IsColumnGlobalAutoIncrement method

Checks whether the specified column's default is set to global autoincrement.

Syntax

Visual Basic
Public Function **IsColumnGlobalAutoIncrement(** _
  ByVal *columnID* As Integer _
) As Boolean

C#
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.

Return value

True if the column is global autoincrementing, false if it is not global autoincrementing.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "GetColumnPartitionSize method" on page 547
- "DatabaseID property" on page 140
- "ColumnCount property" on page 221

IsColumnNewUUID method

Checks whether the specified column's default is set to a new UUID (System.Guid).

Syntax

Visual Basic

```
Public Function IsColumnNewUUID( _
  ByVal columnID As Integer _
) As Boolean
```

C#

```
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.

Return value

True if the column defaults to a new UUID, false if the column does not default to a new UUID.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "ColumnCount property" on page 221
- Guid

IsColumnNullable method

Checks whether the specified column is nullable.

Syntax

Visual Basic

```
Public Function IsColumnNullable( _
  ByVal columnID As Integer _
) As Boolean
```

C#
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.

Return value

True if the column is nullable, false if it is not nullable.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542
- "ColumnCount property" on page 221

IsInPublication method

Checks whether the table is contained in the named publication.

Syntax

Visual Basic
Public Function **IsInPublication(** _
  ByVal *pubName* As String _
) As Boolean

C#
public bool **IsInPublication(**
  string *pubName*
);

Parameters

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

Return value

True if the table is in the publication, false if the table is not in the publication.

See also

- "ULTableSchema class" on page 542
- "ULTableSchema members" on page 542

ULTransaction class

Represents a SQL transaction. This class cannot be inherited.

Syntax

Visual Basic
Public NotInheritable Class **ULTransaction**
  Inherits DbTransaction

C#
public sealed class **ULTransaction**: DbTransaction

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.

Inherits: System.Data.Common.DbTransaction

Implements: System.Data.IDbTransaction, System.IDisposable

See also

- "ULTransaction members" on page 556
- "BeginTransaction() method" on page 145
- "Transaction property" on page 98
- DbTransaction
- IDbTransaction
- IDisposable

ULTransaction members

Public properties

Member name	Description
"Connection property" on page 557	Returns the connection associated with the transaction.
"IsolationLevel property" on page 558	Returns the isolation level for the transaction.

Public methods

Member name	Description
"Commit method" on page 558	Commits the database transaction.
Dispose (inherited from DbTransaction)	Releases the unmanaged resources used by the DbTransaction.
"Rollback method" on page 559	Rolls back the transaction's outstanding changes to the database.

See also

- "ULTransaction class" on page 556
- "BeginTransaction() method" on page 145
- "Transaction property" on page 98
- DbTransaction
- IDbTransaction
- IDisposable

Connection property

Returns the connection associated with the transaction.

Syntax

Visual Basic
Public Readonly Property **Connection** As ULConnection

C#
public ULConnection **Connection** { get;}

Property value

The ULConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Remarks

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

See also

- "ULTransaction class" on page 556
- "ULTransaction members" on page 556
- "BeginTransaction() method" on page 145
- "ULConnection class" on page 131
- IDbTransaction.Connection
- DbCommand.Connection

IsolationLevel property

Returns the isolation level for the transaction.

Syntax

Visual Basic
Public Overrides Readonly Property **IsolationLevel** As IsolationLevel

C#
public override IsolationLevel **IsolationLevel** { get;}

Property value

One of the System.Data.IsolationLevel values. UltraLite.NET only supports System.Data.IsolationLevel.ReadUncommitted.

See also

- "ULTransaction class" on page 556
- "ULTransaction members" on page 556
- "BeginTransaction() method" on page 145
- IsolationLevel
- IsolationLevel.ReadUncommitted

Commit method

Commits the database transaction.

Syntax

Visual Basic
Public Overrides Sub **Commit()**

C#
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

- "ULTransaction class" on page 556
- "ULTransaction members" on page 556
- "Rollback method" on page 559

Rollback method

Rolls back the transaction's outstanding changes to the database.

Syntax

Visual Basic
Public Overrides Sub **Rollback()**

C#
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

- "ULTransaction class" on page 556
- "ULTransaction members" on page 556
- "Commit method" on page 558

Index