UltraLite®
.NET Programming

Published: October 2006

Copyright and trademarks

Contents

About This Manual

Subject

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

Audience

This manual 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.

SQL Anywhere documentation

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

The SQL Anywhere documentation

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

- **SQL Anywhere 10 - Introduction** This book introduces SQL Anywhere 10—a 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 10 - Changes and Upgrading** This book describes new features in SQL Anywhere 10 and in previous versions of the software.
- **SQL Anywhere Server - Database Administration** This book covers material related to running, managing, and configuring SQL Anywhere databases. It describes database connections, the database server, database files, security, backup procedures, security, and replication with Replication Server, as well as administration utilities and options.
- **SQL Anywhere Server - SQL Usage** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- **SQL Anywhere Server - SQL Reference** This book provides a complete reference for the SQL language used by SQL Anywhere. It also describes the SQL Anywhere system views and procedures.
- **SQL Anywhere Server - Programming** This book describes how to build and deploy database applications using the C, C++, and Java programming languages, as well as Visual Studio .NET. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by those tools.
- **SQL Anywhere 10 - Error Messages** This book provides a complete listing of SQL Anywhere error messages together with diagnostic information.
- **MobiLink - Getting Started** This manual introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.
- **MobiLink - Server Administration** This manual describes how to set up and administer MobiLink applications.
- **MobiLink - Client Administration** This manual describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases.
- **MobiLink - Server-Initiated Synchronization** This manual describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization or other remote actions from the consolidated database.

- **QAnywhere** This manual describes QAnywhere, which defines a messaging platform for mobile and wireless clients as well as traditional desktop and laptop clients.
- **SQL Remote** This book describes the SQL Remote data replication system for mobile computing, which enables sharing of data between a SQL Anywhere consolidated database and many SQL Anywhere remote databases using an indirect link such as email or file transfer.
- **SQL Anywhere 10 - Context-Sensitive Help** This manual provides context-sensitive help for the Connect dialog, the Query Editor, the MobiLink Monitor, the SQL Anywhere Console utility, the Index Consultant, and Interactive SQL.
- **UltraLite - Database Management and Reference** This manual introduces the UltraLite database system for small devices.
- **UltraLite - AppForge Programming** This manual describes UltraLite for AppForge. With UltraLite for AppForge you can develop and deploy database applications to handheld, mobile, or embedded devices, running Palm OS, Symbian OS, or Windows CE.
- **UltraLite - .NET Programming** This manual describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.
- **UltraLite - M-Business Anywhere Programming** This manual describes UltraLite for M-Business Anywhere. With UltraLite for M-Business Anywhere you can develop and deploy web-based database applications to handheld, mobile, or embedded devices, running Palm OS, Windows CE, or Windows XP.
- **UltraLite - C and C++ Programming** This manual describes UltraLite C and C++ programming interfaces. With UltraLite you can develop and deploy database applications to handheld, mobile, or embedded devices.

Documentation formats

SQL Anywhere provides documentation in the following formats:

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

  To access the online documentation on Windows operating systems, choose Start ► Programs ► SQL Anywhere 10 ► Online Books. You can navigate the online documentation using the HTML Help table of contents, index, and search facility in the left pane, as well as using the links and menus in the right pane.

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

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

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

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

Documentation conventions

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

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

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

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

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

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

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

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

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

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

  RELEASE SAVEPOINT [*savepoint-name*]

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

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

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

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

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

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

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

File name conventions

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

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

  ```
  MobiLink\redirector
  ```

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

  ```
  MobiLink/redirector
  ```

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

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

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

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

  For more information on the default location of *install-dir*, by operating system, see "File Locations and Installation Settings" [*SQL Anywhere Server - Database Administration*].

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

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

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

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

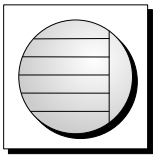
Unix, Linux, and Mac OS X environment variables are stored in shell and login startup files, such as *.cshrc* or *.tcshrc*.

Graphic icons

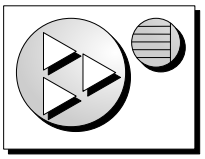
The following icons are used in this documentation.

- A client application.

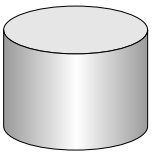
- A database server, such as SQL Anywhere.



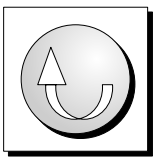
- An UltraLite application.



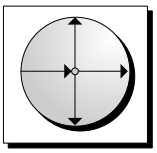
- A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



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



- A Sybase Replication Server



- A programming interface.

Finding out more and providing feedback

Finding out more

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

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

When you write to one of these newsgroups, always provide detailed information about your problem, including the build number of your version of SQL Anywhere. You can find this information by entering **dbeng10 -v** at a command prompt.

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

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

Newsgroup disclaimer
iAnywhere Solutions has no obligation to provide solutions, information, or ideas on its newsgroups, nor is iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and ensure its operation and availability.
iAnywhere Solutions Technical Advisors as well as other staff assist on the newsgroup service when they have time available. They offer their help on a volunteer basis and may not be available on a regular basis to provide solutions and information. Their ability to help is based on their workload.

Feedback

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

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

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

CHAPTER 1

Introduction to UltraLite.NET

Contents

About this chapter

This chapter introduces you to UltraLite.NET. It assumes that you are familiar with the features of UltraLite, as described in "Introducing UltraLite" [*UltraLite - Database Management and Reference*].

☞ For more information about creating applications using UltraLite.NET, see "Understanding UltraLite.NET Development" on page 9.

☞ For a hands-on tutorial introducing UltraLite.NET, see "Tutorial: Build an UltraLite.NET Application" on page 31.

UltraLite and .NET

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

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

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.

System requirements and supported platforms

Development platforms

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

- Microsoft Windows XP.
- Visual Studio.NET, Visual Studio.NET 2003, or Visual Studio 2005.
- For Windows CE devices, .NET Compact Framework version 1.0.5000 or later.

Target platforms

UltraLite.NET supports the following target platforms:

- Microsoft .NET Compact Framework 2.0 and .NET Framework 2.0.
- Microsoft .NET Compact Framework 1.0.3705 or later on Windows CE 3.0 and higher, with Pocket PC on ARM processors. The Pocket PC 2002 emulator is also supported.
- For Windows CE devices, Microsoft .NET Compact Framework version 1.0.5000 or later.
- Microsoft .NET Compact Framework version 1.0.5000 or later on Windows XP.

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

- **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.
- **ulnet10.dll** This file contains the UltraLite runtime. A separate version of the runtime is provided for each target platform.

☞ For more information, see UltraLite Deployment Option for SQL Anywhere.

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, see "iAnywhere.Data.UltraLite namespace (.NET 1.0)" on page 47 for the ADO.NET interface.

CHAPTER 2

SQL Anywhere Explorer for UltraLite

Contents

About this chapter

This chapter describes the SQL Anywhere Explorer, a component that lets you connect to SQL Anywhere and UltraLite databases from Visual Studio .NET.

Using the SQL Anywhere Explorer in UltraLite projects

In Visual Studio .NET 2003 and 2005, 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 .NET 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 manually as follows:

- Ensure Visual Studio is not running.
- For Visual Studio .NET 2003, run *install-dir\ultralite\UltraLite.NET\assembly\v1\installULNet.exe*.

  For Visual Studio 2005, run *install-dir\ultralite\UltraLite.NET\assembly\v2\installULNet.exe*.

Working with UltraLite database connections in Visual Studio .NET

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 .NET

1. Open the SQL Anywhere Explorer by choosing View ► SQL Anywhere Explorer.
2. In the SQL Anywhere Explorer window, right-click Data Connections, and choose Add Connection.

   The Add Connection dialog appears.
3. Select UltraLite, and then click OK.

   The Connection Properties dialog appears.
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 .NET**

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 choose Delete.

   The connection is removed from the SQL Anywhere Explorer window.

Configuring the SQL Anywhere Explorer

The Visual Studio options dialog 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.

   The Options dialog appears.
2. In the left pane of the Options dialog, 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.

   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 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 .NET, when you drag certain database objects from the SQL Anywhere Explorer and drop them onto Visual Studio .NET 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 .NET.

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 .NET 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 to which you want to add a data component.
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 .NET.

♦ To view a table in Visual Studio .NET

1. Connect to an UltraLite database using the SQL Anywhere Explorer.
2. In the SQL Anywhere Explorer dialog, 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 .NET.

CHAPTER 3

Understanding UltraLite.NET Development

Contents

About this chapter

This chapter explains how to develop applications using UltraLite.NET.

☞ For a hands-on tutorial, see "Tutorial: Build an UltraLite.NET Application" on page 31.

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.

Using SQL Anywhere tools in Visual Studio .NET

Some SQL Anywhere tools are incorporated into Visual Studio .NET 2003 and 2005 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, as well as 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 .NET" on page 6
- "Adding UltraLite database objects using the SQL Anywhere Explorer" on page 8

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.

  For more information, see "Transaction processing in UltraLite" on page 24.

- **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.

  For more information, see "Authenticating users" on page 27.

- **Synchronization** A set of objects governing synchronization is accessed from the Connection object.

  For more information, see "Synchronization in UltraLite applications" on page 28.

- **Tables** UltraLite tables are accessed using methods of the Connection object.

  For more information, see "Accessing and manipulating data with the Table API" on page 18.

- **Commands** A set of objects is provided to handle the execution of dynamic SQL statements and to navigate result sets.

  For more information, see "Accessing and manipulating data using SQL" on page 14.

☞ For more information, see "ULConnection class" on page 81.

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();
   ```


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.

☞ For more information, see "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.

☞ For more information, see "ULConnection class" on page 81 and "ULConnectionParms class" on page 111.

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 creation-time database properties" [*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 18.

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 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.

☞ For more information, see "ULCommand class" on page 54

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.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.

```
conn.Transaction.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.add("", newValue);
cmd.add("", oldValue);
```

4. Execute the statement.

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

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

```
conn.Transaction.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.add("", deleteValue);
```

4. Execute the statement.

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

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


```
conn.Transaction.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 console window.

   ```
   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 console window.

```
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 information about SQL, see "Accessing and manipulating data using SQL" on page 14.

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 a number of 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();
```

☞ For more information, see "ULTable class" on page 389 and "ULTableSchema class" on page 410.

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. For more information, see "Converting data types" [*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.

☞ For more information, see "ULTable class" on page 389.

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. For more information, see "Transaction processing in UltraLite" on page 24.

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();
```


Transaction processing in UltraLite

UltraLite provides transaction processing to ensure the integrity of the data in your database. A transaction is a logical unit of work. Either 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 commited or rolled back, the connection will revert to AutoCommit mode until a subsequent call to ULConnection.BeginTransaction.

Some SQL statements—especially statements that alter the stucture 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.

☞ For more information, see "ULConnection class" on page 81 and "ULTransaction class" on page 423.

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, as well as global properties such as the format of dates and times.

  To obtain a ULDatabaseSchema object, access ULConnection.Schema.

  For more information, see "ULConnection class" on page 81.

- **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 are comprised of schema only, there is no Publication object.

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

☞ For more information, see "ULTableSchema class" on page 410.

Error handling

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 SQL Anywhere 10 - Error Messages [*SQL Anywhere 10 - Error Messages*].

After synchronization, you can use the SyncResult property of the connection to obtain more detailed error information.

☞ For more information, see the following:

- "ULSyncProgressListener interface" on page 379
- "ULSyncResult class" on page 383

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.

☞ For more information, see "Authenticating users" on page 27.

♦ 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.

   ☞ For more information, see "ULConnection class" on page 81.

♦ 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 a more detailed explanation of synchronization, see "UltraLite Clients" [*MobiLink - Client Administration*].

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 10 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-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.
See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].

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 ActiveSync provider 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 50.

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.

CHAPTER 4

Tutorial: Build an UltraLite.NET Application

Contents

About this chapter

This chapter provides a tutorial to guide you through the process of building an UltraLite.NET application using Microsoft Visual Studio.

Introduction

This tutorial guides you through the process of building an UltraLite.NET 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 .NET 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 .NET programming language.
- You have Microsoft Visual Studio installed on your computer and you are familiar with using Visual Studio. This tutorial was developed using Visual Studio .NET 2003 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 UltraLite in Sybase Central.

  For more information, see "Creating an UltraLite database from Sybase Central" [*UltraLite - Database Management and Reference*].

Goals

The goal for the tutorial is to gain competence and familiarity with the process of developing UltraLite.NET 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 as follows:

- Ensure Visual Studio is not running.
- For Visual Studio .NET 2003, run *installULNet.exe* from the folder named *install-dir\ultralite\UltraLite.NET\assembly\v1* .

  For Visual Studio 2005, run *installULNet.exe* from the folder named *install-dir\ultralite\UltraLite.NET\assembly\v2* .

Lesson 1: Create a Visual Studio project

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

♦ To create a Visual Studio project

1. Create a Visual Studio project.

   - From the Visual Studio File menu, choose New ► Project to create a new project.

     The New Project window appears.

   - In the left pane, select either the Visual Basic Projects folder or the Visual C# Projects folder.

     In the right pane, select a Smart Device Application 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.

     The Smart Device Application wizard appears.

   - Choose Pocket PC as the target platform, and select Windows Application as the project type. Click OK.

     A Design workspace appears, displaying a form named Form1.

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.

        The Add Reference window appears.

     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* subdirectory of your SQL Anywhere installation. Select *iAnywhere.Data.UltraLite.dll* and click Open.

     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 language. 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. Choose All Files from the Files of Type drop down list so that DLL files appear.

c. Open the folder corresponding to the processor of the CE device you are using. For Pocket PC emulator in Visual Studio .NET 2003 open the x86 folder, but for Visual Studio 2005 open the ARM folder. Select *ulnet10.dll*; *do not click Open.*

d. Click the arrow on the Open button and select **Link File** to link it to your project.

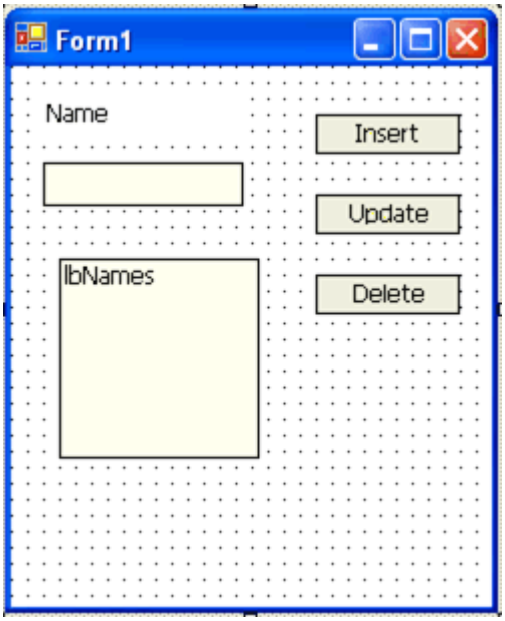
- If you are using Visual Studio 2005, set the Build Action property of the dll to Content and set the Copy to Output Directory Folder property to Copy if Newer.

3. Create a form for your application.

   If the Visual Studio toolbox panel is not currently displayed, choose View ► Toolbox from the main menu. 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	Name	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.NET 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.

      This action deploys your application to the device or emulator, and starts it. The application is deployed to *\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.

   d. From the Debug menu, choose Stop Debugging to close the application.

Lesson 2: Create an UltraLite database

The following procedure creates an UltraLite database using Sybase Central.

☞ For information on using Sybase Central to create an UltraLite database, see "Creating an UltraLite database from Sybase Central" [*UltraLite - Database Management and Reference*].

♦ To create a database

1. Start Sybase Central:

   From the Start menu, choose Programs ▶ SQL Anywhere 10 ▶ Sybase Central.

2. Use the UltraLite plug-in in Sybase Central to create a database in the same directory as your application with the following characteristics:

   - **Database file name** *VBApp.udb* or *CSApp.udb*, depending on your application.
   - **Collation sequence** Use the default collation.
   - **Case sensitivity** Leave this option cleared.
   - **Table name** Type **Names**.
   - **Columns** Create columns in the Names table with the following attributes:

Column Name	Data Type (Size)	Allow NULL?	Default value
ID	integer	No	global autoincrement
Name	varchar(30)	No	None

   - **Primary key** Specify the ID column, ascending.

3. Link the initialized (empty) database file to your 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.
   - Browse to the directory where you created the database file and select the file *VBApp.udb* or *CSApp.udb* depending on your application. *Do not click Open.*
   - Click the arrow in the Open button and click Link File.
   - 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; if you are using Visual Studio 2005, set the Copy to Output Directory property to Copy Always.

Lesson 3: Connect to the database

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

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 created a directory with a different name, use that directory throughout the tutorial.

♦ To add an UltraLite connection to your application

1. Open the source code for the form.

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

2. 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 a Visual C# application, 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 project, add the following code after the declaration of the form components (FriendsWithEvents declarations) and before the Form1 method declaration.

   ```
   '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:

- Opens a connection to the database using the connection parameters set in the ulConnectionParms1 control.
- Calls the RefreshListBox method, which you will define later in this tutorial.
- If an error occurs, prints the error message. For SQL errors, the code also prints the error code.

  For more information about the error code, see SQL Anywhere 10 - Error Messages [*SQL Anywhere 10 - 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 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 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.

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.

☞ For more information, see "Accessing and manipulating data with the Table API" on page 18.

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.

     For more information see SQL Anywhere 10 - Error Messages [*SQL Anywhere 10 - 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. Double-click the Insert button 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.

     For more information about the error code, see SQL Anywhere 10 - Error Messages [*SQL Anywhere 10 - Error Messages*].

   For C#, add the following code to the 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 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. Double-click the Update button 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.

     For more information about the error code, see SQL Anywhere 10 - Error Messages [*SQL Anywhere 10 - Error Messages*].

   For C#, add the following code to the 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 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. Double-click the Delete button 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.

     For more information about the error code, see SQL Anywhere 10 - Error Messages [*SQL Anywhere 10 - Error Messages*].

   For C#, add the following code to the 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 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.

♦ 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 *ulnet10.dll* that you included in your application.

3. Choose Debug ► Start.

   This builds an executable file containing your application and deploys it to the Pocket PC 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. For more information, see "Lesson 3: Connect to the database" on page 37.
- Confirm that you chose Link File when adding the database file to the project and you set the Build Action to Content. If using Visual Studio 2005, confirm that you set Copy to Output Directory to Copy Allways or Copy if Newer. 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 *ulnet10.dll* for your target platform, or ran the Windows CE 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. For more information, see "Lesson 1: Create a 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 listbox. Click Delete. The name should no longer appear in the listbox.

This completes the tutorial.

CHAPTER 5

iAnywhere.Data.UltraLite namespace (.NET 1.0)

Contents

About this chapter

The **iAnywhere.Data.UltraLite** namespace contains the classes, interfaces, and enumerations of the UltraLite.NET Data Provider for ADO.NET.

UltraLite.NET allows you to write C# or Visual Basic .NET code to develop UltraLite database applications using the ADO.NET standard and provides a migration path to SQL Anywhere. If you are undecided as to whether to use UltraLite or SQL Anywhere, start with the iAnywhere.Data.UltraLite namespace and switch to SQL Anywhere (iAnywhere.Data.SQLAnywhere namespace) if you determine that more advanced SQL Anywhere features are required. If you may be moving to SQL Anywhere, avoid as much as possible the few UltraLite-only extensions that are in the iAnywhere.Data.UltraLite namespace.

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 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.

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: System.Globalization.CultureInfo.CurrentUICulture, then System.Globalization.CultureInfo.CurrentCulture, and finally culture "EN".

ULActiveSyncListener interface

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

Prototypes

' Visual Basic

Public Interface **ULActiveSyncListener**

// C#

public interface **ULActiveSyncListener**

ULActiveSyncListener members

Public methods

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

ActiveSyncInvoked method

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

Prototypes

' 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 SignalSyncIsComplete method 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();
  }
}
```

ULAuthStatusCode enumeration

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

Prototypes

' Visual Basic

```
Public Enum ULAuthStatusCode
  Inherits Short
```

// C#

```
public enum ULAuthStatusCode :
  short
```

Member name

Member name	Description
EXPIRED	User ID or password has expired - authorization failed (EXPIRED = 3).
IN_USE	User ID is already in use - authorization failed (IN_USE = 5).
INVALID	Bad user ID or password - authorization failed (INVALID = 4).
UNKNOWN	Authorization status is unknown, possibly because the connection has not yet performed a synchronization (UNKNOWN = 0).
VALID	User ID and password were valid at time of synchronization (VALID = 1).
VALID_BUT_EXPIRES_SOON	User ID and password were valid at time of synchronization, but will expire soon (VALID_BUT_EXPIRES_SOON = 2).

See also

- "AuthStatus property" on page 384

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.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULCommand
  Inherits Component
```

// C#

```
public sealed class ULCommand :
  Component
```

Remarks

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

The Transaction property 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
ExecuteNonQuery method	Executes a statement that does not return a result set, such as a SQL INSERT, DELETE, or UPDATE statement.
ExecuteReader method	Executes a SQL SELECT statement and returns the result set in a ULDataReader class. Use this method for creating read-only result sets.
ExecuteResultSet method	**UL Ext.:** Executes a SQL SELECT statement and returns the result set in a ULResultSet class. Use this method for creating mutable result sets.
ExecuteScalar method	Executes a SQL SELECT statement and returns a single value.
ExecuteTable method	**UL Ext.:** Retrieves a database table in a ULTable class for direct manipulation. The CommandText property is interpreted as the name of the table and the IndexName property can be used to specify a table sorting order. The CommandType property must be CommandType.TableDirect.

You can reset most properties, including the CommandText property, 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 or explicitly call the System.ComponentModel.Component.Dispose. In Visual Basic, you always explicitly call the System.ComponentModel.Component.Dispose.

Implements: IDbCommand, IDisposable

ULCommand members

Public constructors

Member name	Description
ULCommand constructor	Initializes a ULCommand object.
ULCommand constructor	Initializes a ULCommand object with the specified command text.
ULCommand constructor	Initializes a ULCommand object with the specified command text and connection.
ULCommand constructor	Initializes a ULCommand object with the specified command text, connection, and transaction.

Public properties

Member name	Description
CommandText property	Specifies the text of the SQL statement or the name of the table when the CommandType property is CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.
CommandTimeout property	This feature is not supported by UltraLite.NET.
CommandType property	Specifies the type of command to be executed.
Connection property	The connection object on which to execute the ULCommand object.
DesignTimeVisible property	Indicates if the ULCommand should be visible in a customized Windows Form Designer control.
IndexName property	**UL Ext.:** Specifies the name of the index to open (sort) the table with when the CommandType property is CommandType.TableDirect.
Parameters property	Specifies the parameters for the current statement.
Plan property	**UL Ext.:** Returns the access plan UltraLite.NET uses to execute a query. This property is intended primarily for use during development.
Transaction property	Specifies the ULTransaction class in which the ULCommand executes.
UpdatedRowSource property	Specifies how command results are applied to the DataRow when used by the Update method of the ULDataAdapter.

Public methods

Member name	Description
Cancel method	This method is not supported in UltraLite.NET.
CreateParameter method	Provides a ULParameter class object for supplying parameters to ULCommand objects.
ExecuteNonQuery method	Executes a statement that does not return a result set, such as a SQL INSERT, DELETE, or UPDATE statement.
ExecuteReader method	Executes a SQL SELECT statement and returns the result set.
ExecuteReader method	Executes a SQL SELECT statement with the specified command behavior and returns the result set.
ExecuteResultSet method	**UL Ext.:** Executes a SQL SELECT statement and returns the result set as a ULResultSet class.
ExecuteResultSet method	**UL Ext.:** Executes a SQL SELECT statement with the specified command behavior and returns the result set as a ULResultSet class.
ExecuteScalar method	Executes a SQL SELECT statement and returns a single value.
ExecuteTable method	**UL Ext.:** Retrieves in a ULTable class a database table for direct manipulation. The CommandText property is interpreted as the name of the table and IndexName property can be used to specify a table sorting order.
ExecuteTable method	**UL Ext.:** Retrieves, with the specified command behavior, a database table for direct manipulation. The CommandText property is interpreted as the name of the table and IndexName property can be used to specify a table sorting order.
Prepare method	Pre-compiles and stores the SQL statement of this command.

ULCommand constructor

Initializes a ULCommand object.

Prototypes

' Visual Basic

Overloads Public Sub **New()**

// C#

public **ULCommand();**

Remarks

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

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "CreateCommand method" on page 97
- "ULCommand constructor" on page 57
- "ULCommand constructor" on page 58
- "ULCommand constructor" on page 58

ULCommand constructor

Initializes a ULCommand object with the specified command text.

Prototypes

```
' Visual Basic

Overloads 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 CommandType property is CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.

Remarks

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

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "CreateCommand method" on page 97
- "ULCommand constructor" on page 56
- "ULCommand constructor" on page 58
- "ULCommand constructor" on page 58

ULCommand constructor

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

Prototypes

' Visual Basic

```
Overloads 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 CommandType property is CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.
- **connection** The ULConnection class object representing the current connection.

Remarks

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

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "CreateCommand method" on page 97
- "ULCommand constructor" on page 56
- "ULCommand constructor" on page 57
- "ULCommand constructor" on page 58

ULCommand constructor

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

Prototypes

' Visual Basic

```
Overloads 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 CommandType property is CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.
- **connection** The ULConnection class object representing the current connection.
- **transaction** The ULTransaction class in which the ULCommand executes.

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "CreateCommand method" on page 97
- "ULCommand constructor" on page 56
- "ULCommand constructor" on page 57
- "ULCommand constructor" on page 58

CommandText property

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

Prototypes

```
' Visual Basic

NotOverridable Public Property CommandText As String _
  Implements IDbCommand.CommandText

// C#

public 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

It is recommended that SELECT statements used to create read-only result sets (ExecuteReader method or ExecuteScalar method) should end with " FOR READ ONLY". For some statements that use temporary tables, there may be a significant performance improvement.

Implements

IDbCommand.CommandText

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 54
- "ULCommand members" on page 55
- "ExecuteNonQuery method" on page 66
- "ExecuteReader method" on page 66
- "ExecuteResultSet method" on page 68
- "ExecuteScalar method" on page 70
- "ExecuteTable method" on page 71

CommandTimeout property

This feature is not supported by UltraLite.NET.

Prototypes

' Visual Basic

NotOverridable Public Property **CommandTimeout** As Integer _
  Implements IDbCommand.CommandTimeout

// C#

public int **CommandTimeout** {get;set;}

Property value

The value is always zero.

Exceptions

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

Implements

IDbCommand.CommandTimeout

CommandType property

Specifies the type of command to be executed.

Prototypes

' Visual Basic

```
NotOverridable Public Property CommandType As CommandType _
  Implements IDbCommand.CommandType
```

// C#

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

Property value

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

Remarks

Supported command types are as follows:

- CommandType.TableDirect - **UL Ext.:** When you specify this CommandType, the CommandText property must be the name of a database table. You can also specify the index used to open (sort) the table with IndexName property. Use ExecuteTable method or ExecuteReader method to access the table.
- CommandType.Text - When you specify this CommandType, the CommandText property must be a SQL statement or query. Use ExecuteNonQuery method to execute a non-query SQL statement and use either ExecuteReader method or ExecuteScalar method to execute a query.

Exceptions

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

Implements

IDbCommand.CommandType

Connection property

The connection object on which to execute the ULCommand object.

Prototypes

' Visual Basic

```
Public Property Connection As ULConnection
```

// C#

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

Property value

The ULConnection class 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 IDbCommand.Connection.

DesignTimeVisible property

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

Prototypes

' Visual Basic

Public Property **DesignTimeVisible** As Boolean

// C#

public 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.

IndexName property

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

Prototypes

' 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 54
- "ULCommand members" on page 55
- "ExecuteTable method" on page 71
- "ExecuteReader method" on page 66

Parameters property

Specifies the parameters for the current statement.

Prototypes

' Visual Basic

Public Readonly Property **Parameters** As ULParameterCollection

// C#

public ULParameterCollection **Parameters** {get;}

Property value

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

Remarks

Use question marks in the CommandText property 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 CommandText property as there are parameters in this collection.

This is the strongly-typed version of IDbCommand.Parameters.

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "ULParameter class" on page 256

Plan property

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

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

Transaction property

Specifies the ULTransaction class in which the ULCommand executes.

Prototypes

' Visual Basic

```
Public Property Transaction As ULTransaction
```

// C#

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

Property value

The ULTransaction class in which the ULCommand executes. This should be the current transaction of the connection specified by the Connection property. 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 IDbCommand.Transaction.

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "BeginTransaction method" on page 92

UpdatedRowSource property

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

Prototypes

' Visual Basic

```
NotOverridable Public Property UpdatedRowSource As UpdateRowSource _
  Implements IDbCommand.UpdatedRowSource
```

// C#

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

Property value

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

Implements

IDbCommand.UpdatedRowSource

Cancel method

This method is not supported in UltraLite.NET.

Prototypes

' Visual Basic

```
NotOverridable Public Sub Cancel() _
  Implements IDbCommand.Cancel
```

// C#

```
public void Cancel();
```

Remarks

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

Implements

IDbCommand.Cancel

CreateParameter method

Provides a ULParameter class object for supplying parameters to ULCommand objects.

Prototypes

' Visual Basic

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

// C#

```
public ULParameter CreateParameter();
```

Return value

A new parameter, as a ULParameter class 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 class object. You can set properties on the ULParameter to specify the value for the parameter.

This is the strongly-typed version of IDbCommand.CreateParameter.

ExecuteNonQuery method

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

Prototypes

' Visual Basic

```
NotOverridable Public Function ExecuteNonQuery() As Integer _
  Implements IDbCommand.ExecuteNonQuery
```

// C#

```
public int ExecuteNonQuery();
```

Return value

The number of rows affected.

Remarks

The statement is the current ULCommand object, with the CommandText property and Parameters property 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 CommandType property cannot be CommandType.TableDirect.

Exceptions

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

Implements

IDbCommand.ExecuteNonQuery

ExecuteReader method

Executes a SQL SELECT statement and returns the result set.

Prototypes

' Visual Basic

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

// C#

```
public ULDataReader ExecuteReader();
```


Return value

The result set as a ULDataReader class object.

Remarks

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

If the CommandType property is CommandType.TableDirect, ExecuteReader performs an ExecuteTable method and returns a ULTable class downcast as a ULDataReader class.

It is recommended that SELECT statements used with this method to create read-only result sets end with FOR READ ONLY. For some statements that use temporary tables, there may be a significant performance improvement.

This is the strongly-typed version of IDbCommand.ExecuteReader.

Exceptions

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

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "ExecuteReader method" on page 67

ExecuteReader method

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

Prototypes

' Visual Basic

```
Overloads Public Function ExecuteReader( _
   ByVal cmdBehavior As CommandBehavior _
) As ULDataReader
```

// C#

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


Parameters

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

Return value

The result set as a ULDataReader class object.

Remarks

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

If the CommandType property is CommandType.TableDirect, ExecuteReader performs an ExecuteTable method and returns a ULTable class downcast as a ULDataReader class.

It is recommended that SELECT statements used with this method to create read-only result sets end with FOR READ ONLY. For some statements that use temporary tables, there may be a significant performance improvement.

This is the strongly-typed version of IDbCommand.ExecuteReader.

Exceptions

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

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "ExecuteReader method" on page 66

ExecuteResultSet method

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

Prototypes

' Visual Basic

Overloads Public Function **ExecuteResultSet()** As ULResultSet

// C#

public ULResultSet **ExecuteResultSet();**

Return value

The result set as a ULResultSet class object.

Remarks

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

If the CommandType property is CommandType.TableDirect, ExecuteReader performs an ExecuteTable method and returns a ULTable class downcast as a ULResultSet class.

Exceptions

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

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "ExecuteResultSet method" on page 69

ExecuteResultSet method

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

Prototypes

```
' Visual Basic

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

// C#

public ULResultSet ExecuteResultSet(
  CommandBehavior cmdBehavior
);
```

Parameters

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

Return value

The result set as a ULResultSet class object.

Remarks

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

If the CommandType property is CommandType.TableDirect, ExecuteReader performs an ExecuteTable method and returns a ULTable class downcast as a ULResultSet class.

Exceptions

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

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "ExecuteReader method" on page 66

ExecuteScalar method

Executes a SQL SELECT statement and returns a single value.

Prototypes

' Visual Basic

```
NotOverridable Public Function ExecuteScalar() As Object _
  Implements IDbCommand.ExecuteScalar
```

// C#

```
public 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 CommandText property and any Parameters property 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 CommandType property is CommandType.TableDirect, ExecuteScalar performs an ExecuteTable method and returns the first column of the first row.

It is recommended that SELECT statements used with this method to create read-only result sets end with FOR READ ONLY. For some statements that use temporary tables, there may be a significant performance improvement.

Exceptions

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

Implements

IDbCommand.ExecuteScalar

ExecuteTable method

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

Prototypes

' Visual Basic

Overloads Public Function **ExecuteTable()** As ULTable

// C#

public ULTable **ExecuteTable();**

Return value

The table as a ULTable class object.

Remarks

The CommandType property must be set to CommandType.TableDirect.

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

Exceptions

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

See also

- "ULCommand class" on page 54
- "ULCommand members" on page 55
- "ExecuteTable method" on page 72

ExecuteTable method

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

Prototypes

```
' Visual Basic

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

// C#

public ULTable ExecuteTable(
  CommandBehavior cmdBehavior
);
```

Parameters

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

Return value

The table as a ULTable class object.

Remarks

The CommandType property must be set to CommandType.TableDirect.

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

Exceptions

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

See also

- "ULCommand class" on page 54

- "ULCommand members" on page 55
- "ExecuteTable method" on page 71

Prepare method

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

Prototypes

' Visual Basic

```
NotOverridable Public Sub Prepare() _
  Implements IDbCommand.Prepare
```

// C#

```
public 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.

Exceptions

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

Implements

IDbCommand.Prepare

ULCommandBuilder class

Automatically generates single-table commands used to reconcile changes made to aDataSet with the associated database.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULCommandBuilder
  Inherits Component
```

// C#

```
public sealed class ULCommandBuilder :
  Component
```

Remarks

The ULDataAdapter class does not automatically generate the SQL statements required to reconcile changes made to a 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 class. Then, any additional SQL statements that you do not set are generated by the ULCommandBuilder.

Inherits: System.ComponentModel.Component

Implements: IDisposable

ULCommandBuilder members

Public constructors

Member name	Description
ULCommandBuilder constructor	Initializes a ULCommandBuilder object.
ULCommandBuilder constructor	Initializes a ULCommandBuilder object with the specified ULDataAdapter class object.

Public properties

Member name	Description
DataAdapter property	Gets or sets a ULDataAdapter class object for which SQL statements are automatically generated.
QuotePrefix property	Gets or sets the starting character or characters to use when specifying UltraLite database objects, such as tables or columns, whose names contain characters such as spaces or reserved tokens.

Member name	Description
QuoteSuffix property	Gets or sets the ending character or characters to use when specifying UltraLite database objects, such as tables or columns, whose names contain characters such as spaces or reserved tokens.

Public methods

Member name	Description
GetDeleteCommand method	Gets the automatically generated ULCommand class object required to perform deletions on the database.
GetInsertCommand method	Gets the automatically generated ULCommand class object required to perform insertions on the database.
GetUpdateCommand method	Gets the automatically generated ULCommand class object required to perform updates on the database.
RefreshSchema method	Clears the commands associated with this ULCommandBuilder.

ULCommandBuilder constructor

Initializes a ULCommandBuilder object.

Prototypes

' Visual Basic

Overloads Public Sub **New()**

// C#

public **ULCommandBuilder();**

See also

- "ULCommandBuilder class" on page 74
- "ULCommandBuilder members" on page 74
- "ULCommandBuilder constructor" on page 75

ULCommandBuilder constructor

Initializes a ULCommandBuilder object with the specified ULDataAdapter class object.

Prototypes

' Visual Basic

Overloads Public Sub **New(** _
  ByVal *adapter* As ULDataAdapter _
)

// C#

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

Parameters

- **adapter** A ULDataAdapter class object.

DataAdapter property

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

Prototypes

' Visual Basic

```
Public Property DataAdapter As ULDataAdapter
```

// C#

```
public ULDataAdapter DataAdapter {get;set;}
```

Property value

A ULDataAdapter class object.

QuotePrefix property

Gets or sets the starting character or characters to use when specifying UltraLite database objects, such as tables or columns, whose names contain characters such as spaces or reserved tokens.

Prototypes

' Visual Basic

```
Public Property QuotePrefix As String
```

// C#

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

Property value

The starting character or characters to use. The default is an empty string.

See also

- "ULCommandBuilder class" on page 74
- "ULCommandBuilder members" on page 74
- "QuoteSuffix property" on page 77

QuoteSuffix property

Gets or sets the ending character or characters to use when specifying UltraLite database objects, such as tables or columns, whose names contain characters such as spaces or reserved tokens.

Prototypes

' Visual Basic

Public Property **QuoteSuffix** As String

// C#

public string **QuoteSuffix** {get;set;}

Property value

The ending character or characters to use. The default is an empty string.

See also

- "ULCommandBuilder class" on page 74
- "ULCommandBuilder members" on page 74
- "QuotePrefix property" on page 76

GetDeleteCommand method

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

Prototypes

' Visual Basic

Public Function **GetDeleteCommand()** As ULCommand

// C#

public ULCommand **GetDeleteCommand();**

Return value

The automatically generated ULCommand class object required to perform deletions.

Remarks

After the SQL statement is first generated, the application must explicitly call the RefreshSchema method if it changes the SelectCommand property in any way. Otherwise, the GetDeleteCommand 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 Update method or the GetDeleteCommand method.

Exceptions

- InvalidOperationException - The DataAdapter property has not been initialized.

  The DataAdapter.SelectCommand property has not been initialized.

The DataAdapter.SelectCommand.Connection property has not been initialized.

Dynamic SQL generation is not supported against multiple base tables.

Dynamic SQL generation is not supported against a SelectCommand that contains duplicate columns.

Dynamic SQL generation for the DeleteCommand is not supported against a SelectCommand that does not return any key column information.

GetInsertCommand method

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

Prototypes

' Visual Basic

Public Function **GetInsertCommand()** As ULCommand

// C#

public ULCommand **GetInsertCommand();**

Return value

The automatically generated ULCommand class object required to perform insertions.

Remarks

After the SQL statement is first generated, the application must explicitly call the RefreshSchema method if it changes the SelectCommand property in any way. Otherwise, the GetInsertCommand 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 Update method or the GetInsertCommand method.

Exceptions

- InvalidOperationException - The DataAdapter property has not been initialized.

  The DataAdapter.SelectCommand property has not been initialized.

  The DataAdapter.SelectCommand.Connection property has not been initialized.

  Dynamic SQL generation for the InsertCommand is not supported against a SelectCommand that does not return any modifiable columns.

  Dynamic SQL generation is not supported against multiple base tables.

  Dynamic SQL generation is not supported against a SelectCommand that contains duplicate columns.

GetUpdateCommand method

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

Prototypes

' Visual Basic

Public Function **GetUpdateCommand()** As ULCommand

// C#

public ULCommand **GetUpdateCommand();**

Return value

The automatically generated ULCommand class object required to perform updates.

Remarks

After the SQL statement is first generated, the application must explicitly call the RefreshSchema method if it changes the SelectCommand property in any way. Otherwise, the GetUpdateCommand 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 Update method or the GetUpdateCommand method.

Exceptions

- InvalidOperationException - The DataAdapter property has not been initialized.

  The DataAdapter.SelectCommand property has not been initialized.

  The DataAdapter.SelectCommand.Connection property has not been initialized.

  Dynamic SQL generation for the UpdateCommand is not supported against a SelectCommand that does not return any modifiable columns.

  Dynamic SQL generation is not supported against multiple base tables.

  Dynamic SQL generation is not supported against a SelectCommand that contains duplicate columns.

  Dynamic SQL generation for the UpdateCommand is not supported against a SelectCommand that does not return any key column information.

RefreshSchema method

Clears the commands associated with this ULCommandBuilder.

Prototypes

' Visual Basic

Public Sub **RefreshSchema()**


```
// C#

public void RefreshSchema();
```

Remarks

After the SQL statement is first generated, the application must explicitly call RefreshSchema method if it changes the SelectCommand propertystatement in any way. Otherwise, GetInsertCommand method, GetDeleteCommand method, and 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 Update method, GetInsertCommand method, GetDeleteCommand method, or GetUpdateCommand method.

ULConnection class

Represents a connection to an UltraLite.NET database.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULConnection
  Inherits Component
```

// C#

```
public sealed class ULConnection :
  Component
```

Remarks

To use the UltraLite Engine runtime of UltraLite.NET, set RuntimeType property to the appropriate value before using any other UltraLite.NET API.

A connection to an existing database is opened using the Open method.

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 Schema property.

Implements: IDbConnection, IDisposable

ULConnection members

Public static fields (shared)

Member name	Description
INVALID_DATABASE_ID field	**UL Ext.:** A database ID constant indicating that the DatabaseID property has not been set.

Public static properties (shared)

Member name	Description
DatabaseManager property	**UL Ext.:** Provides access to the singleton ULDatabaseManager class object.

Public constructors

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

Public properties

Member name	Description
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 class object.
ConnectionTimeout property	This feature is not supported by UltraLite.NET.
Database property	Returns the name of the database to which the connection opens.
DatabaseID property	**UL Ext.:** Specifies the Database ID value to be used for global autoincrement columns.
GlobalAutoIncrementUsage property	**UL Ext.:** Returns the percentage of available global autoincrement values that have been used.
LastIdentity property	**UL Ext.:** Returns the most recent identity value used.
Schema property	**UL Ext.:** Provides access to the schema of the current database associated with this connection.
State property	Returns the current state of the connection.
SyncParms property	**UL Ext.:** Specifies the synchronization settings for this connection.
SyncResult property	**UL Ext.:** Returns the results of the last synchronization for this connection.

Public methods

Member name	Description
BeginTransaction method	Returns a transaction object. Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a Commit method or Rollback method.
BeginTransaction 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 a Commit method or Rollback method.

Member name	Description
ChangeDatabase method	Changes the current database for an open ULConnection.
ChangeEncryptionKey method	**UL Ext.:** Changes the database's encryption key to the specified new key.
Close method	Closes the database connection.
CountUploadRows method	**UL Ext.:** Returns the number of rows that need to be uploaded when the next synchronization takes place.
CountUploadRows method	**UL Ext.:** Returns the number of rows that need to be uploaded when the next synchronization takes place.
CreateCommand method	Creates and initializes a ULCommand class object associated with this connection and its current transaction. You can use the properties of the ULCommand to control its behavior.
ExecuteTable method	**UL Ext.:** Retrieves in a ULTable class a database table for direct manipulation. The table is opened (sorted) using the table's primary key.
ExecuteTable method	**UL Ext.:** Retrieves in a ULTable class a database table for direct manipulation. The table is opened (sorted) using the specified index.
ExecuteTable method	**UL Ext.:** Retrieves, with the specified command behavior, a database table for direct manipulation. The table is opened (sorted) using the specified index.
GetLastDownloadTime method	**UL Ext.:** Returns the time of the most recent download of the specified publication.
GetNewUUID method	**UL Ext.:** Generates a new UUID (Guid).
GrantConnectTo method	**UL Ext.:** Grants access to an UltraLite database for a user ID with a specified password.
Open method	Opens a connection to a database using the previously-specified connection string.
ResetLastDownloadTime method	**UL Ext.:** Resets the time of the most recent download.
RevokeConnectFrom method	**UL Ext.:** Revokes access to an UltraLite database from the specified user ID.
RollbackPartialDownload method	**UL Ext.:** Rolls back outstanding changes to the database from a partial download.
StartSynchronizationDelete method	**UL Ext.:** Marks all subsequent deletes made by this connection for synchronization.
StopSynchronizationDelete method	**UL Ext.:** Prevents delete operations from being synchronized.

Member name	Description
Synchronize method	**UL Ext.:** Synchronize the database using the current SyncParms property.
Synchronize method	**UL Ext.:** Synchronize the database using the current SyncParms property with progress events posted to the specified listener.

Public events

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

ULConnection constructor

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

Prototypes

' Visual Basic

Overloads Public Sub **New()**

// C#

public **ULConnection();**

Remarks

To use the UltraLite Engine runtime of UltraLite.NET, set RuntimeType property to the appropriate value before using any other UltraLite.NET API.

The ULConnection object needs to have the ConnectionString property set before it can be opened.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "Open method" on page 104
- "ULConnection constructor" on page 84

ULConnection constructor

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

Prototypes

```
' Visual Basic

Overloads 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 the ConnectionString property.

Remarks

To use the UltraLite Engine runtime of UltraLite.NET, set RuntimeType property to the appropriate value before using any other UltraLite.NET API.

The connection string can be supplied using a ULConnectionParms class object.

Exceptions

- ArgumentException - The supplied connection string is invalid.

Example

The following code creates and opens a connection to the existing database \UltraLite\MyDatabase.udb on a Windows CE 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 81
- "ULConnection members" on page 81
- "Open method" on page 104
- "ULConnection constructor" on page 84

INVALID_DATABASE_ID field

UL Ext.: A database ID constant indicating that the DatabaseID property has not been set.

Prototypes

' Visual Basic

```
Public Shared INVALID_DATABASE_ID As Long
```

// C#

```
public const long INVALID_DATABASE_ID;
```

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 class object.

Prototypes

' Visual Basic

```
NotOverridable Public Property ConnectionString As String _
  Implements IDbConnection.ConnectionString
```

// C#

```
public 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 String Parameters Reference" [*UltraLite - Database Management and Reference*].

Parameter values must not contain semi-colons (;), or begin with either a single quote (') or a double quote (") character. Leading and trailling 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 method) and remove the DBA user (using RevokeConnectFrom method).

Exceptions

- ArgumentException - The supplied connection string is invalid.
- InvalidOperationException - The value cannot be set while the connection is open.

Implements

IDbConnection.ConnectionString

Example

The following code creates and opens a connection to the existing database \UltraLite\MyDatabase.udb on a Windows CE 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 81
- "ULConnection members" on page 81
- "Open method" on page 104

ConnectionTimeout property

This feature is not supported by UltraLite.NET.

Prototypes

' Visual Basic

NotOverridable Public Readonly Property **ConnectionTimeout** As Integer _
  Implements IDbConnection.ConnectionTimeout

// C#

public int **ConnectionTimeout** {get;}

Property value

The value is always zero.

Exceptions

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

Implements

IDbConnection.ConnectionTimeout

Database property

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

Prototypes

' Visual Basic

```
NotOverridable Public Readonly Property Database As String _
  Implements IDbConnection.Database
```

// C#

```
public string  Database {get;}
```

Property value

A string containing the name of the database.

Remarks

On Windows CE 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.

Implements

IDbConnection.Database

DatabaseID property

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

Prototypes

' 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,UInt32.MaxValue]. A value of INVALID_DATABASE_ID field is used to indicate that the database ID has not been set for the current database.

Exceptions

- ULException class - The specified new database ID is invalid.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "GetDatabaseProperty method" on page 171
- "SetDatabaseOption method" on page 176

DatabaseManager property

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

Prototypes

' Visual Basic

Public Shared Readonly Property **DatabaseManager** As ULDatabaseManager

// C#

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

Property value

A reference to the singleton ULDatabaseManager class object.

GlobalAutoIncrementUsage property

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

Prototypes

' 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 DatabaseID property.

Exceptions

- ULException class - A SQL error occurred.

LastIdentity property

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

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

Schema property

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

Prototypes

' Visual Basic

Public Readonly Property **Schema** As ULDatabaseSchema

// C#

public ULDatabaseSchema **Schema** {get;}

Property value

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

Remarks

This property is only valid while its connection is open.

State property

Returns the current state of the connection.

Prototypes

' Visual Basic

```
NotOverridable Public Readonly Property State As ConnectionState _
  Implements IDbConnection.State
```

// C#

```
public ConnectionState State {get;}
```

Property value

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

Implements

IDbConnection.State

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "StateChange event" on page 109

SyncParms property

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

Prototypes

' Visual Basic

```
Public Readonly Property SyncParms As ULSyncParms
```

// C#

```
public ULSyncParms SyncParms {get;}
```

Property value

A reference to the ULSyncParms class 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 81

- "ULConnection members" on page 81
- "Synchronize method" on page 107
- "SyncResult property" on page 92

SyncResult property

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

Prototypes

' Visual Basic

Public Readonly Property **SyncResult** As ULSyncResult

// C#

public ULSyncResult **SyncResult** {get;}

Property value

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

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "Synchronize method" on page 107
- "SyncParms property" on page 91

BeginTransaction method

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

Prototypes

' Visual Basic

Overloads Public Function **BeginTransaction()** As ULTransaction

// C#

public ULTransaction **BeginTransaction();**

Return value

A ULTransaction class object representing the new transaction.

Remarks

To associate a command with a transaction object, use the Transaction property. The current transaction is automatically associated to commands created by CreateCommand method.

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 until the next call to BeginTransaction.

This is the strongly-typed version of IDbConnection.BeginTransaction.

Exceptions

- ULException class - The connection is closed.
- InvalidOperationException - ULConnection does not support parallel transactions.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "BeginTransaction method" on page 93

BeginTransaction 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 a Commit method or Rollback method.

Prototypes

' Visual Basic

```
Overloads 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 IsolationLevel.ReadUncommitted.

Return value

A ULTransaction class object representing the new transaction.

Remarks

To associate a command with a transaction object, use the Transaction property. The current transaction is automatically associated to commands created by CreateCommand method.

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 until the next call to BeginTransaction.

This is the strongly-typed version of IDbConnection.BeginTransaction.

Exceptions

- ULException class - The connection is closed or an unsupported isolation level was specified.
- InvalidOperationException - ULConnection does not support parallel transactions.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "BeginTransaction method" on page 92

ChangeDatabase method

Changes the current database for an open ULConnection.

Prototypes

```
' Visual Basic

NotOverridable Public Sub ChangeDatabase( _
   ByVal connectionString As String _
) _
   Implements IDbConnection.ChangeDatabase

// C#

public 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.

Implements

IDbConnection.ChangeDatabase

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "ConnectionString property" on page 86

ChangeEncryptionKey method

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

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULConnection class” on page 81
- “ULConnection members” on page 81
- “EncryptionKey property” on page 117

Close method

Closes the database connection.

Prototypes

' Visual Basic

```
NotOverridable Public Sub Close() _
  Implements IDbConnection.Close
```

// C#

```
public void Close();
```

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

Implements

IDbConnection.Close

CountUploadRows method

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

Prototypes

' Visual Basic

```
Overloads Public Function CountUploadRows( _
  ByVal mask As Integer, _
  ByVal threshold As UInt32 _
) As UInt32
```

// C#

```
public uint CountUploadRows(
  int mask,
  uint threshold
);
```

Parameters

- **mask** The set of publications to check for rows. For more information, see the ULPublicationSchema class.
- **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).

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "CountUploadRows method" on page 96

CountUploadRows method

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

Prototypes

' Visual Basic

```
Overloads Public Function CountUploadRows( _
  ByVal mask As Integer, _
  ByVal threshold As Long _
) As Long
```

// C#

```
public long CountUploadRows(
  int mask,
  long threshold
);
```

Parameters

- **mask** The set of publications to check for rows. For more information, see the ULPublicationSchema class.
- **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 [0,0x0fffffff].

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.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULConnection class” on page 81
- “ULConnection members” on page 81
- “CountUploadRows method” on page 96

CreateCommand method

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

Prototypes

' Visual Basic

```
Public Function CreateCommand() As ULCommand
```



```
// C#

public ULCommand CreateCommand();
```

Return value

A new ULCommand class object.

Remarks

You must set the CommandText property before the command can be executed.

This is the strongly-typed version of IDbConnection.CreateCommand.

ExecuteTable method

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

Prototypes

```
' Visual Basic

Overloads 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 class object.

Remarks

This method is a shortcut for the ExecuteTable method that does not require a ULCommand class 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()).

Exceptions

- ULException class - A SQL error occurred.
- InvalidOperationException - The *tableName* is invalid.

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 81
- "ULConnection members" on page 81
- "ExecuteTable method" on page 99
- "ExecuteTable method" on page 100

ExecuteTable method

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

Prototypes

' Visual Basic

```
Overloads 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 class object.

Remarks

This method is a shortcut for the ExecuteTable method that does not require a ULCommand class 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()).

Exceptions

- ULException class - A SQL error occurred.
- InvalidOperationException - The *tableName* is invalid.

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 81
- “ULConnection members” on page 81
- “ExecuteTable method” on page 98
- “ExecuteTable method” on page 100

ExecuteTable method

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

Prototypes

' Visual Basic

Overloads 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 CommandBehavior flags describing the results of the query and its effect on the connection. UltraLite.NET respects only the CommandBehavior.Default, CommandBehavior.CloseConnection, and CommandBehavior.SchemaOnly flags.

Return value

The table as a ULTable class object.

Remarks

This method is a shortcut for the ExecuteTable method that does not require a ULCommand class 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()).

Exceptions

- ULException class - A SQL error occurred.
- InvalidOperationException - The *tableName* is invalid.

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 81
- "ULConnection members" on page 81
- "ExecuteTable method" on page 98
- "ExecuteTable method" on page 99

GetLastDownloadTime method

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

Prototypes

' Visual Basic

```
Public Function GetLastDownloadTime( _
  ByVal mask As Integer _
) As Date
```

// C#

```
public DateTime GetLastDownloadTime(
  int mask
);
```

Parameters

- **mask** The mask of the publication to check. For more information, see the ULPublicationSchema class.

Return value

The timestamp of the last download.

Remarks

The parameter *mask* must reference a single publication or be the special constant SYNC_ALL_DB field for the time of the last download of the full database.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "ResetLastDownloadTime method" on page 105

GetNewUUID method

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

Prototypes

' Visual Basic

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

// C#

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

Return value

A new UUID as a Guid.

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

GrantConnectTo method

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

Prototypes

' 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 81
- "ULConnection members" on page 81
- "UserID property" on page 118
- "Password property" on page 118
- "ConnectionString property" on page 86

Open method

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

Prototypes

' Visual Basic

```
NotOverridable Public Sub Open() _
  Implements IDbConnection.Open
```

// C#

```
public void Open();
```

Remarks

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

Exceptions

- ULException class - A SQL error occurred while attempting to open the database.
- InvalidOperationException - The connection is already open or the connection string is not specified in the ConnectionString property.

Implements

IDbConnection.Open

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "ConnectionString property" on page 86
- "State property" on page 91

ResetLastDownloadTime method

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

Prototypes

' Visual Basic

```
Public Sub ResetLastDownloadTime( _
  ByVal mask As Integer _
)
```

// C#

```
public void ResetLastDownloadTime(
  int mask
);
```

Parameters

- **mask** The set of publications to reset. For more information, see the ULPublicationSchema class.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "GetLastDownloadTime method" on page 102

RevokeConnectFrom method

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

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "GrantConnectTo method" on page 103

RollbackPartialDownload method

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

Prototypes

' Visual Basic

Public Sub **RollbackPartialDownload()**

// C#

public void **RollbackPartialDownload();**

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "KeepPartialDownload property" on page 360
- "ResumePartialDownload property" on page 363

StartSynchronizationDelete method

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

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULConnection class” on page 81
- “ULConnection members” on page 81
- “StopSynchronizationDelete method” on page 107
- “Truncate method” on page 408

StopSynchronizationDelete method

UL Ext.: Prevents delete operations from being synchronized.

Prototypes

' Visual Basic

Public Sub **StopSynchronizationDelete()**

// C#

public void **StopSynchronizationDelete();**

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULConnection class” on page 81
- “ULConnection members” on page 81
- “StartSynchronizationDelete method” on page 106

Synchronize method

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

Prototypes

' Visual Basic

Overloads Public Sub **Synchronize()**

// C#

public void **Synchronize();**

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "Synchronize method" on page 108

Synchronize method

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

Prototypes

```
' Visual Basic

Overloads Public Sub Synchronize( _
  ByVal listener As ULSyncProgressListener _
)

// C#

public void Synchronize(
  ULSyncProgressListener listener
);
```

Parameters

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

Remarks

The last event posted to the listener will have a state of ULSyncProgressState enumeration.

Errors during synchronization will be posted as ULSyncProgressState enumeration events and then thrown as ULException classs.

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

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 81
- "ULConnection members" on page 81
- "ULSyncProgressListener interface" on page 379
- "Synchronize method" on page 107

InfoMessage event

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

Prototypes

' 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 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);
```

StateChange event

Occurs when this connection changes state.

Prototypes

' Visual Basic

Public Event **StateChange** As StateChangeEventHandler

// C#

public event StateChangeEventHandler **StateChange**;

Remarks

To process state change messages, you must create a 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);
```


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.

Prototypes

' 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 (Open method) or dropping a database (DropDatabase method).

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 ToString method. The resulting string is used to create a new ULConnection class with the ULConnection constructor constructor or set the ConnectionString property of an existing ULConnection class 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 CE, 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 CE 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 informaton, see the AdditionalParms property.

Optional connection settings

Depending on your application's needs and how the database was created, you might need to supply a non-default UserID property and Password property, a database EncryptionKey property, and the connection CacheSize property. If your application is using multiple connections, you should provide a unique ConnectionName property 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 RevokeConnectFrom method. To add a user or change a user's password, use the GrantConnectTo method.

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 ChangeEncryptionKey method.

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

ULConnectionParms members

Public constructors

Member name	Description
ULConnectionParms constructor	Initializes a ULConnectionParms instance with its default values.

Public properties

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

Public methods

Member name	Description
ToString method	Returns the string representation of this instance.

Public events

Member name	Description
UnusedEvent event	Unused.

ULConnectionParms constructor

Initializes a ULConnectionParms instance with its default values.

Prototypes

' Visual Basic

Public Sub **New()**

// C#

public **ULConnectionParms();**

AdditionalParms property

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

Prototypes

' 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 ConnectionString property. 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 filename 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 filename parameter (DatabaseOnCE property or DatabaseOnDesktop property). This parameter is required if the application (or UltraLite engine) needs to access two different databases that have the same base filename. This parameter is only used when opening a connection with Open method.
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, as well as 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 Open method.
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. See the RuntimeType property for more information on using the UltraLite engine with UltraLite.NET.

Exceptions

- ArgumentException - The value contained an invalid connection string.

CacheSize property

Specifies the size of the cache.

Prototypes

' 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"
```

The default cache size is 16 pages. Using the default page size of 4 KB, the default cache size is therefore 64 KB. The minimum cache size is platform dependent.

The default cache size is conservative. If your testing shows the need for better performance, you should increase the cache size.

Increasing the cache size beyond the size of the database itself provides no performance improvement and large cache sizes might interfere with the number of other applications you can use.

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

Exceptions

- ArgumentException - The value contained a semicolon (;), or began with either a single quote (') or a double quote (").

ConnectionName property

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

Prototypes

' 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).

Exceptions

- ArgumentException - The value contained a semicolon (;), or began with either a single quote (') or a double quote (").

DatabaseOnCE property

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

Prototypes

' 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).

Exceptions

- ArgumentException - The value contained a semicolon (;), or began with either a single quote (') or a double quote (").

DatabaseOnDesktop property

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

Prototypes

' 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).

Exceptions

- ArgumentException - The value contained a semicolon (;), or began with either a single quote (') or a double quote (").

EncryptionKey property

Specifies a key for encrypting the database.

Prototypes

' 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.

Exceptions

- ArgumentException - The value contained a semicolon (;), or began with either a single quote (') or a double quote (").

See also

- “ULConnectionParms class” on page 111
- “ULConnectionParms members” on page 112
- “ChangeEncryptionKey method” on page 95

Password property

Specifies the password for the authenticated user.

Prototypes

' 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.

Exceptions

- ArgumentException - The value contained a semicolon (;), or began with either a single quote (') or a double quote (").

See also

- "ULConnectionParms class" on page 111
- "ULConnectionParms members" on page 112
- "UserID property" on page 118

UserID property

Specifies an authenticated user for the database.

Prototypes

' 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 GrantConnectTo method) and remove the DBA user (using RevokeConnectFrom method).

Exceptions

- ArgumentException - The value contained a semicolon (;), or began with either a single quote (') or a double quote (").

See also

- "ULConnectionParms class" on page 111
- "ULConnectionParms members" on page 112
- "Password property" on page 118
- "GrantConnectTo method" on page 103
- "RevokeConnectFrom method" on page 105

ToString method

Returns the string representation of this instance.

Prototypes

' Visual Basic

Overrides Public 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.

UnusedEvent event

Unused.

Prototypes

' 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.

ULConnectionParms.UnusedEventHandler delegate

UL Ext.: Unused.

Prototypes

' Visual Basic

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

// C#

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

Parameters

- **sender** Object that is the sender.
- **args** Event arguments.

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.

ULCreateParms class

UL Ext.: Builds a string of creation-time options for creating an UltraLite database.

Prototypes

' Visual Basic

Public Class **ULCreateParms**

// C#

public class **ULCreateParms**

Remarks

A ULCreateParms object is used to specify the parameters for creating a database (CreateDatabase method).

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 ToString method. The resulting string can then be used as the createParms parameter of the CreateDatabase method.

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

See also

- "ULCreateParms members" on page 122
- "GetDatabaseProperty method" on page 171

ULCreateParms members

Public constructors

Member name	Description
ULCreateParms constructor	Initializes a ULCreateParms instance with its default values.

Public properties

Member name	Description
CaseSensitive property	Specifies whether the new database should be case sensitive when comparing string values.
ChecksumLevel property	Specifies the level of database page checksums enabled for the new database.

Member name	Description
DateFormat property	Specifies the date format used for string conversions by the new database.
DateOrder property	Specifies the date order used for string conversions by the new database.
FIPS property	Specifies whether the new database should be using AES_FIPS encryption or AES encryption.
MaxHashSize property	Specifies the default maximum number of bytes to use for index hashing in the new database.
NearestCentury property	Specifies the nearest century used for string conversions by the new database.
Obfuscate property	Specifies whether the new database should be using obfuscation (simple encryption) or not.
PageSize property	Specifies the page size of the new database, in bytes or kilobytes.
Precision property	Specifies the floating-point precision used for string conversions by the new database.
Scale property	Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION during string conversions by the new database.
TimeFormat property	Specifies the time format used for string conversions by the new database.
TimestampFormat property	Specifies the timestamp format used for string conversions by the new database.
TimestampIncrement property	Specifies the minimum difference between two unique timestamps, in nanoseconds (1,000,000th of a second).
UTF8Encoding property	Specifies whether the new database should be using the UTF8 character set or the character set associated with the collation.

Public methods

Member name	Description
ToString method	Returns the string representation of this instance.

ULCreateParms constructor

Initializes a ULCreateParms instance with its default values.

Prototypes

' Visual Basic

Public Sub **New()**

// C#

public **ULCreateParms();**

CaseSensitive property

Specifies whether the new database should be case sensitive when comparing string values.

Prototypes

' 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.

ChecksumLevel property

Specifies the level of database page checksums enabled for the new database.

Prototypes

' 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.

Exceptions

- ArgumentException - The value is invalid.

DateFormat property

Specifies the date format used for string conversions by the new database.

Prototypes

' 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).

Exceptions

- ArgumentException - The value contained a semicolon (;), or began with either a single quote (') or a double quote (").

DateOrder property

Specifies the date order used for string conversions by the new database.

Prototypes

' Visual Basic

```
Public Property DateOrder As ULDateOrder
```

// C#

```
public ULDateOrder DateOrder {get;set;}
```

Property value

A ULDateOrder enumeration value identifying the date order for string conversions. The default is YMD.

FIPS property

Specifies whether the new database should be using AES_FIPS encryption or AES encryption.

Prototypes

' 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 (see the EncryptionKey property) when the new database is created. If FIPS is set true and no encryption key is supplied, the CreateDatabase method will fail with a missing encryption key error.

MaxHashSize property

Specifies the default maximum number of bytes to use for index hashing in the new database.

Prototypes

' 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.

Exceptions

- ArgumentException - The value is invalid.

NearestCentury property

Specifies the nearest century used for string conversions by the new database.

Prototypes

' 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.

Exceptions

- ArgumentException - The value is invalid.

Obfuscate property

Specifies whether the new database should be using obfuscation (simple encryption) or not.

Prototypes

' 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 (FIPS property). 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.

PageSize property

Specifies the page size of the new database, in bytes or kilobytes.

Prototypes

' 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.

Exceptions

- ArgumentException - The value is invalid.

Precision property

Specifies the floating-point precision used for string conversions by the new database.

Prototypes

' Visual Basic

Public Property **Precision** As Integer

// C#

public int **Precision** {get;set;}

Property value

An integer specifying the precison. The value must be in the range [1,127]. The default is 30.

Exceptions

- ArgumentException - The value is invalid.

See also

- “ULCreateParms class” on page 122
- “ULCreateParms members” on page 122
- “Scale property” on page 128

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.

Prototypes

' 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.

Exceptions

- ArgumentException - The value is invalid.

TimeFormat property

Specifies the time format used for string conversions by the new database.

Prototypes

' 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).

Exceptions

- ArgumentException - The value contained a semicolon (;), or began with either a single quote (') or a double quote (").

TimestampFormat property

Specifies the timestamp format used for string conversions by the new database.

Prototypes

' 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).

Exceptions

- ArgumentException - The value contained a semicolon (;), or began with either a single quote (') or a double quote (").

TimestampIncrement property

Specifies the minimum difference between two unique timestamps, in nanoseconds (1,000,000th of a second).

Prototypes

' 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.

Exceptions

- ArgumentException - The value is invalid.

UTF8Encoding property

Specifies whether the new database should be using the UTF8 character set or the character set associated with the collation.

Prototypes

' 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 wish 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.

ToString method

Returns the string representation of this instance.

Prototypes

' Visual Basic

Overrides Public 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.

ULCursorSchema class

UL Ext.: Represents the schema of an UltraLite.NET cursor.

Prototypes

' 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.

ULCursorSchema members

Public properties

Member name	Description
ColumnCount property	Returns the number of columns in the cursor.
IsOpen property	Checks whether the cursor schema is currently open.
Name property	Returns the name of the cursor.

Public methods

Member name	Description
GetColumnID method	Returns the column ID of the named column.
GetColumnName method	Returns the name of the column identified by the specified column ID.
GetColumnPrecision method	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
GetColumnScale method	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
GetColumnSize method	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
GetColumnSQLName method	Returns the name of the column identified by the specified column ID.

Member name	Description
GetColumnULDbType method	Returns the UltraLite.NET data type of the column identified by the specified column ID.
GetSchemaTable method	Returns a DataTable that describes the column schema of the UL-DataReader class.

Protected methods

Member name	Description
Finalize method	Releases the unmanaged resources used by the ULCursorSchema.

ColumnCount property

Returns the number of columns in the cursor.

Prototypes

' 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.

IsOpen property

Checks whether the cursor schema is currently open.

Prototypes

' 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.

Name property

Returns the name of the cursor.

Prototypes

' Visual Basic

```
Public Readonly Property Name As String
```

// C#

```
public string  Name {get;}
```

Property value

The name of the cursor as a string.

Finalize method

Releases the unmanaged resources used by the ULCursorSchema.

Prototypes

' Visual Basic

```
Overrides Protected Sub Finalize()
```

// C#

```
protected override void Finalize();
```

GetColumnID method

Returns the column ID of the named column.

Prototypes

' 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 property-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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULCursorSchema class" on page 132
- "ULCursorSchema members" on page 132
- "ColumnCount property" on page 133

GetColumnName method

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

Prototypes

```
' 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 property-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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULCursorSchema class" on page 132
- "ULCursorSchema members" on page 132
- "ColumnCount property" on page 133

GetColumnPrecision method

Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).

Prototypes

' 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 property-1].

Return value

The precision of the specified numeric column.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULCursorSchema class" on page 132
- "ULCursorSchema members" on page 132
- "GetColumnULDbType method" on page 139

GetColumnScale method

Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).

Prototypes

' 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 property-1].

Return value

The scale of the specified numeric column.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULCursorSchema class" on page 132
- "ULCursorSchema members" on page 132
- "GetColumnULDbType method" on page 139

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).

Prototypes

' 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 property-1].

Return value

The size of the specified sized column.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULCursorSchema class" on page 132
- "ULCursorSchema members" on page 132
- "GetColumnULDbType method" on page 139

GetColumnSQLName method

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

Prototypes

```
' 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 property-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 method 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 likey 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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULCursorSchema class" on page 132
- "ULCursorSchema members" on page 132
- "ColumnCount property" on page 133

GetColumnULDbType method

Returns the UltraLite.NET data type of the column identified by the specified column ID.

Prototypes

```
' 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 property-1].

Return value

A ULDbType enumeration enumerated integer.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULCursorSchema class" on page 132
- "ULCursorSchema members" on page 132
- "ColumnCount property" on page 133

GetSchemaTable method

Returns a DataTable that describes the column schema of the ULDataReader class.

Prototypes

' Visual Basic

Public Function **GetSchemaTable()** As DataTable

// C#

public DataTable **GetSchemaTable();**

Return value

A DataTable that describes the column schema.

Remarks

For more information, see GetSchemaTable method.

ULDataAdapter class

Represents a set of commands and a database connection used to fill a DataSet and to update a database.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULDataAdapter
  Inherits Component
```

// C#

```
public sealed class ULDataAdapter :
  Component
```

Remarks

The DataSet provides a way to work with data offline; that is, away from your UltraLite database. The ULDataAdapter provides methods to associate a 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 class or the ULTable class provide more efficient access to data.

Implements: IDbDataAdapter, IDataAdapter, IDisposable

ULDataAdapter members

Public constructors

Member name	Description
ULDataAdapter constructor	Initializes a ULDataAdapter object.
ULDataAdapter constructor	Initializes a ULDataAdapter object with the specified SELECT statement.
ULDataAdapter constructor	Initializes a ULDataAdapter object with the specified SELECT statement and connection.
ULDataAdapter constructor	Initializes a ULDataAdapter object with the specified SELECT statement and connection string.

Public properties

Member name	Description
AcceptChangesDuringFill property	Specifies whether DataRow.AcceptChanges is called on a DataRow after it is added to the DataTable.

Member name	Description
ContinueUpdateOnError property	Specifies whether to generate an exception when an error is encountered during a row update.
DeleteCommand property	Specifies a ULCommand class object that is executed against the database when DbDataAdapter.Update is called to delete rows in the database that correspond to deleted rows in the DataSet.
InsertCommand property	Specifies a ULCommand class object that is executed against the database when DbDataAdapter.Update is called to insert rows in the database that correspond to inserted rows in the DataSet.
MissingMappingAction property	Determines the action to take when incoming data does not have a matching table or column.
MissingSchemaAction property	Determines the action to take when the existing DataSet schema does not match incoming data.
SelectCommand property	Specifies a ULCommand class that is used during Fill method or FillSchema method to obtain a result set from the database for copying into a DataSet.
TableMappings property	Returns a collection that provides the master mapping between a source table and a DataTable
UpdateCommand property	Specifies a ULCommand class object that is executed against the database when Update method is called to update rows in the database that correspond to updated rows in the DataSet.

Public methods

Member name	Description
Fill method	Adds or refreshes rows into the DataTable named "Table" of the specified DataSet. The DataTable named "Table" is created and added to the DataSet if necessary.
Fill method	Adds or refreshes rows into the named DataTable of the specified DataSet. The DataTable named is created and added to the DataSet if neccessary.
Fill method	Adds or refreshes the specified range of rows into the named DataTable of the specified DataSet. The DataTable named is created and added to the DataSet if neccessary.
Fill method	Adds or refreshes rows into the specified DataTable.
FillSchema method	Adds a DataTable named "Table" to a DataSet and configures the schema to match the schema in the data source.
FillSchema method	Adds a DataTable named "Table" to a DataSet and configures the schema to match the schema in the data source.

Member name	Description
FillSchema method	Configures the schema of a DataTable to match the schema in the data source.
GetFillParameters method	Returns the parameters set by the user when executing a SELECT statement.
Update method	Updates the rows in the database with the changes made to the DataTable named "Table" of the specified DataSet.
Update method	Updates the rows in the database with the changes made to the named DataTable of the specified DataSet.
Update method	Updates the rows in the database with the changes made to the specified DataTable.
Update method	Updates the rows in the database with the changes made to the specified DataRow array.

Public events

Member name	Description
FillError event	Occurs when an error is detected during a fill operation.
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.
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.

ULDataAdapter constructor

Initializes a ULDataAdapter object.

Prototypes

' Visual Basic

Overloads Public Sub **New()**

// C#

public **ULDataAdapter();**

See also

- “ULDataAdapter class” on page 141
- “ULDataAdapter members” on page 141
- “ULDataAdapter constructor” on page 144
- “ULDataAdapter constructor” on page 144
- “ULDataAdapter constructor” on page 145

ULDataAdapter constructor

Initializes a ULDataAdapter object with the specified SELECT statement.

Prototypes

' Visual Basic

```
Overloads Public Sub New( _
  ByVal selectCommand As ULCommand _
)
```

// C#

```
public ULDataAdapter(
  ULCommand selectCommand
);
```

Parameters

- **selectCommand** A ULCommand class object that is used during DbDataAdapter.Fill to select records from the data source for placement in the DataSet.

See also

- "ULDataAdapter class" on page 141
- "ULDataAdapter members" on page 141
- "ULDataAdapter constructor" on page 143
- "ULDataAdapter constructor" on page 144
- "ULDataAdapter constructor" on page 145

ULDataAdapter constructor

Initializes a ULDataAdapter object with the specified SELECT statement and connection.

Prototypes

' Visual Basic

```
Overloads 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 SelectCommand property of the ULDataAdapter.

- **selectConnection** A ULConnection class object that defines a connection to a database.

See also

- "ULDataAdapter class" on page 141
- "ULDataAdapter members" on page 141
- "ULDataAdapter constructor" on page 143
- "ULDataAdapter constructor" on page 144
- "ULDataAdapter constructor" on page 145

ULDataAdapter constructor

Initializes a ULDataAdapter object with the specified SELECT statement and connection string.

Prototypes

```
' Visual Basic

Overloads 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 SelectCommand property of the ULDataAdapter.
- **selectConnectionString** A connection string for an UltraLite.NET database.

See also

- "ULDataAdapter class" on page 141
- "ULDataAdapter members" on page 141
- "ULDataAdapter constructor" on page 143
- "ULDataAdapter constructor" on page 144
- "ULDataAdapter constructor" on page 144

AcceptChangesDuringFill property

Specifies whether DataRow.AcceptChanges is called on a DataRow after it is added to the DataTable.

Prototypes

' Visual Basic

Public Property **AcceptChangesDuringFill** As Boolean

// C#

public bool **AcceptChangesDuringFill** {get;set;}

Property value

True to specify that the ULDataAdapter should call the DataRow.AcceptChanges function on the DataRow; false if AcceptChanges is not to be called, and the newly added rows are treated as inserted rows. The default is true.

ContinueUpdateOnError property

Specifies whether to generate an exception when an error is encountered during a row update.

Prototypes

' Visual Basic

Public Property **ContinueUpdateOnError** As Boolean

// C#

public bool **ContinueUpdateOnError** {get;set;}

Property value

True to continue the update without generating an exception, false to generate an exception. The default is false.

Remarks

If ContinueUpdateOnError is true, no exception is thrown when an error occurs during the update of a row. The update of the row is skipped and the error information is placed in the DataRow.RowError of the row. The ULDataAdapter continues to update subsequent rows.

If ContinueUpdateOnError is false, an exception is thrown when an error occurs.

DeleteCommand property

Specifies a ULCommand class object that is executed against the database when DbDataAdapter.Update is called to delete rows in the database that correspond to deleted rows in the DataSet.

Prototypes

' Visual Basic

Public Property **DeleteCommand** As ULCommand

// C#

public ULCommand **DeleteCommand** {get;set;}

Property value

A ULCommand class object that is executed to delete rows in the database that correspond to deleted rows in the DataSet.

Remarks

When DeleteCommand is assigned to an existing ULCommand class object, the ULCommand class object is not cloned. The DeleteCommand maintains a reference to the existing ULCommand class.

This is the strongly-typed version of IDbDataAdapter.DeleteCommand .

InsertCommand property

Specifies a ULCommand class object that is executed against the database when DbDataAdapter.Update is called to insert rows in the database that correspond to inserted rows in the DataSet.

Prototypes

' Visual Basic

Public Property **InsertCommand** As ULCommand

// C#

public ULCommand **InsertCommand** {get;set;}

Property value

A ULCommand class object that is executed to insert rows in the database that correspond to inserted rows in the DataSet.

Remarks

When InsertCommand is assigned to an existing ULCommand class object, the ULCommand class object is not cloned. The InsertCommand maintains a reference to the existing ULCommand class.

This is the strongly-typed version of IDbDataAdapter.InsertCommand .

MissingMappingAction property

Determines the action to take when incoming data does not have a matching table or column.

Prototypes

' Visual Basic

NotOverridable Public Property **MissingMappingAction** As MissingMappingAction _
  Implements IDataAdapter.MissingMappingAction

// C#

public MissingMappingAction **MissingMappingAction** {get;set;}

Property value

One of the MissingMappingAction values. The default is MissingMappingAction.Passthrough.

Implements

IDataAdapter.MissingMappingAction

MissingSchemaAction property

Determines the action to take when the existing DataSet schema does not match incoming data.

Prototypes

' Visual Basic

NotOverridable Public Property **MissingSchemaAction** As MissingSchemaAction _
  Implements IDataAdapter.MissingSchemaAction

// C#

public MissingSchemaAction **MissingSchemaAction** {get;set;}

Property value

One of the MissingSchemaAction values. The default is MissingSchemaAction.Add.

Implements

IDataAdapter.MissingSchemaAction

SelectCommand property

Specifies a ULCommand class that is used during Fill method or FillSchema method to obtain a result set from the database for copying into a DataSet.

Prototypes

' Visual Basic

Public Property **SelectCommand** As ULCommand

// C#

public ULCommand **SelectCommand** {get;set;}

Property value

A ULCommand class object that is executed to fill the DataSet.

Remarks

When SelectCommand is assigned to an existing ULCommand class object, the ULCommand class object is not cloned. The SelectCommand maintains a reference to the existing ULCommand class.

If the SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised. The SELECT statement can also be specified in the ULDataAdapter constructor, ULDataAdapter constructor, or ULDataAdapter constructor constructors.

This is the strongly-typed version of IDbDataAdapter.SelectCommand .

TableMappings property

Returns a collection that provides the master mapping between a source table and a DataTable

Prototypes

' Visual Basic

Public Readonly Property **TableMappings** As DataTableMappingCollection

// C#

public DataTableMappingCollection **TableMappings** {get;}

Property value

A collection of DataTableMapping objects providing the master mapping between source tables and DataTables. The default value is an empty collection.

Remarks

When reconciling changes, the ULDataAdapter uses the DataTableMappingCollection collection to associate the column names used by the data source with the column names used by the DataSet.

This is the strongly-typed version of IDataAdapter.TableMappings.

UpdateCommand property

Specifies a ULCommand class object that is executed against the database when Update method is called to update rows in the database that correspond to updated rows in the DataSet.

Prototypes

' Visual Basic

Public Property **UpdateCommand** As ULCommand

// C#

public ULCommand **UpdateCommand** {get;set;}

Property value

A ULCommand class object that is executed to update rows in the database that correspond to updated rows in the DataSet.

Remarks

When UpdateCommand is assigned to an existing ULCommand class object, the ULCommand class object is not cloned. The UpdateCommand maintains a reference to the existing ULCommand class.

If execution of this command returns rows, these rows may be merged with the DataSet depending on how you set the UpdatedRowSource property of the ULCommand class object.

This is the strongly-typed version of IDbDataAdapter.UpdateCommand .

Fill method

Adds or refreshes rows into the DataTable named "Table" of the specified DataSet. The DataTable named "Table" is created and added to the DataSet if necessary.

Prototypes

```
' Visual Basic

Overloads NotOverridable Public Function Fill( _
   ByVal dataSet As DataSet _
) As Integer _
   Implements IDataAdapter.Fill

// C#

public int Fill(
   DataSet dataSet
);
```

Parameters

- **dataSet** A DataSet to fill with records and optionally schema.

Return value

The number of rows successfully added or refreshed in the DataSet.

Remarks

For large result sets, this can have a significant performance impact. An alternative is to use a ULDataReader class when a read-only result set is sufficient, perhaps with SQL statements (ExecuteNonQuery) to carry out modifications. Another alternative is to use a ULTable class that allows read-write access to the database.

If SelectCommand property does not return any rows, no tables are added to the DataSet and no exception is raised.

For more information, see IDataAdapter.Fill.

Exceptions

- System.ArgumentNullException - The *dataSet* parameter is invalid.
- InvalidOperationException - The SelectCommand property is invalid or, the table mapping is missing and MissingMappingAction property is set to MissingMappingAction.Error or, the DataTable is missing from the DataSet and MissingSchemaAction property is set to MissingSchemaAction.Error.

Implements

IDataAdapter.Fill

Fill method

Adds or refreshes rows into the named DataTable of the specified DataSet. The DataTable named is created and added to the DataSet if neccessary.

Prototypes

' Visual Basic

```
Overloads Public Function Fill( _
   ByVal dataSet As DataSet, _
   ByVal srcTable As String _
) As Integer
```

// C#

```
public int Fill(
   DataSet dataSet,
   string  srcTable
);
```

Parameters

- **dataSet** A DataSet to fill with records and optionally schema.
- **srcTable** The name of the source table to use for table mapping.

Return value

The number of rows successfully added or refreshed in the DataSet.

Remarks

For large result sets, this can have a significant performance impact. An alternative is to use an ULDataReader class when a read-only result set is sufficient, perhaps with SQL statements (ExecuteNonQuery) to carry out modifications. Another alternative is to use a ULTable class that allows read-write access to the database.

If SelectCommand property does not return any rows, no tables are added to the DataSet and no exception is raised.

For more information, see IDataAdapter.Fill.

Exceptions

- System.ArgumentNullException - The *dataSet* or *srcTable* parameter is invalid.
- InvalidOperationException - The SelectCommand property is invalid or, the table mapping is missing and MissingMappingAction property is set to MissingMappingAction.Error or, the DataTable is missing from the DataSet and MissingSchemaAction property is set to MissingSchemaAction.Error.

Fill method

Adds or refreshes the specified range of rows into the named DataTable of the specified DataSet. The DataTable named is created and added to the DataSet if neccessary.

Prototypes

' Visual Basic

```
Overloads Public Function Fill( _
   ByVal dataSet As DataSet, _
   ByVal startRecord As Integer, _
   ByVal maxRecords As Integer, _
   ByVal srcTable As String _
) As Integer
```

// C#

```
public int Fill(
   DataSet dataSet,
   int startRecord,
   int maxRecords,
   string  srcTable
);
```

Parameters

- **dataSet** A DataSet to fill with records and optionally schema.
- **startRecord** The zero-based record number with which to start.
- **maxRecords** The maximum number of records to be read into the DataSet. A value of 0 gets all records found after the start record.
- **srcTable** The name of the source table to use for table mapping.

Return value

The number of rows successfully added or refreshed in the DataSet.

Remarks

Even if you use the startRecord argument to limit the number of records that are copied to the DataSet, all records in the ULDataAdapter query are fetched from the database to the client. For large result sets, this can have a significant performance impact. An alternative is to use an ULDataReader class when a read-only result set is sufficient, perhaps with SQL statements (ExecuteNonQuery) to carry out modifications. Another alternative is to use a ULTable class that allows read-write access to the database.

If SelectCommand property does not return any rows, no tables are added to the DataSet and no exception is raised.

For more information, see IDataAdapter.Fill.

Exceptions

- ArgumentException - The *startRecord* or *maxRecords* parameter was less than zero.
- System.ArgumentNullException - The *dataSet* or *srcTable* parameter is invalid.
- InvalidOperationException - The SelectCommand property is invalid or, the table mapping is missing and MissingMappingAction property is set to MissingMappingAction.Error or, the DataTable is missing from the DataSet and MissingSchemaAction property is set to MissingSchemaAction.Error.

Fill method

Adds or refreshes rows into the specified DataTable.

Prototypes

' Visual Basic

```
Overloads Public Function Fill( _
  ByVal dataTable As DataTable _
) As Integer
```

// C#

```
public int Fill(
  DataTable dataTable
);
```

Parameters

- **dataTable** A DataTable to fill with records and optionally schema.

Return value

The number of rows successfully added or refreshed in the DataTable.

Remarks

For large result sets, this can have a significant performance impact. An alternative is to use an ULDataReader class when a read-only result set is sufficient, perhaps with SQL statements (ExecuteNonQuery) to carry out modifications. Another alternative is to use a ULTable class that allows read-write access to the database.

If SelectCommand property does not return any rows, no tables are added and no exception is raised.

For more information, see IDataAdapter.Fill.

Exceptions

- System.ArgumentNullException - The *dataTable* parameter is invalid.

- InvalidOperationException - The SelectCommand property is invalid or, the table mapping is missing and MissingMappingAction property is set to MissingMappingAction.Error.

FillSchema method

Adds a DataTable named "Table" to a DataSet and configures the schema to match the schema in the data source.

Prototypes

```
' Visual Basic

Overloads NotOverridable Public Function FillSchema( _
   ByVal dataSet As DataSet, _
   ByVal schemaType As SchemaType _
) As DataTable() _
   Implements IDataAdapter.FillSchema

// C#

public DataTable[] FillSchema(
   DataSet dataSet,
   SchemaType schemaType
);
```

Parameters

- **dataSet** A DataSet to fill with the schema.
- **schemaType** One of the SchemaType values that specify how to insert the schema.

Return value

A reference to a collection of DataTable objects that were added to the DataSet.

Remarks

For more information, see IDataAdapter.FillSchema.

Exceptions

- System.ArgumentNullException - The *dataSet* parameter is invalid.
- InvalidOperationException - The SelectCommand property is invalid or, the table mapping is missing and MissingMappingAction property is set to MissingMappingAction.Error.

Implements

IDataAdapter.FillSchema

FillSchema method

Adds a DataTable named "Table" to a DataSet and configures the schema to match the schema in the data source.

Prototypes

' Visual Basic

```
Overloads Public Function FillSchema( _
  ByVal dataSet As DataSet, _
  ByVal schemaType As SchemaType, _
  ByVal srcTable As String _
) As DataTable()
```

// C#

```
public DataTable[] FillSchema(
  DataSet dataSet,
  SchemaType schemaType,
  string  srcTable
);
```

Parameters

- **dataSet** A DataSet to fill with the schema.
- **schemaType** One of the SchemaType values that specify how to insert the schema.
- **srcTable** The name of the source table to use for table mapping.

Return value

A reference to a collection of DataTable objects that were added to the DataSet.

Remarks

For more information, see DbDataAdapter.FillSchema.

Exceptions

- System.ArgumentNullException - The *dataSet* or *srcTable* parameter is invalid.
- InvalidOperationException - The SelectCommand property is invalid or, the table mapping is missing and MissingMappingAction property is set to MissingMappingAction.Error.

FillSchema method

Configures the schema of a DataTable to match the schema in the data source.

Prototypes

' Visual Basic

```
Overloads Public Function FillSchema( _
  ByVal dataTable As DataTable, _
  ByVal schemaType As SchemaType _
) As DataTable
```

// C#

```
public DataTable FillSchema(
```



```
  DataTable dataTable,
  SchemaType schemaType
);
```

Parameters

- **dataTable** A DataTable to fill with the schema.
- **schemaType** One of the SchemaType values that specify how to insert the schema.

Return value

A reference to the DataTable object that contains the schema.

Remarks

For more information, see DbDataAdapter.FillSchema.

Exceptions

- System.ArgumentNullException - The *dataTable* parameter is invalid.
- InvalidOperationException - The SelectCommand property is invalid.

GetFillParameters method

Returns the parameters set by the user when executing a SELECT statement.

Prototypes

```
' Visual Basic

Public Function GetFillParameters() As ULParameter()

// C#

public ULParameter[] GetFillParameters();
```

Return value

An array of ULParameter class objects that contains the parameters set by the user.

Update method

Updates the rows in the database with the changes made to the DataTable named "Table" of the specified DataSet.

Prototypes

```
' Visual Basic

Overloads NotOverridable Public Function Update( _
  ByVal dataSet As DataSet _
) As Integer _
  Implements IDataAdapter.Update
```



```
// C#

public int Update(
   DataSet dataSet
);
```

Parameters

- **dataSet** A DataSet to update with records from.

Return value

The number of rows successfully updated from the DataSet.

Remarks

For more information, see IDataAdapter.Update.

Exceptions

- System.ArgumentNullException - The *dataSet* parameter is invalid.
- InvalidOperationException - The SelectCommand property is invalid or, the table mapping is missing and MissingMappingAction property is set to MissingMappingAction.Error or, the DataTable is missing from the DataSet.

Implements

IDataAdapter.Update

Update method

Updates the rows in the database with the changes made to the named DataTable of the specified DataSet.

Prototypes

```
' Visual Basic

Overloads Public Function Update( _
   ByVal dataSet As DataSet, _
   ByVal srcTable As String _
) As Integer
```

```
// C#

public int Update(
   DataSet dataSet,
   string  srcTable
);
```

Parameters

- **dataSet** A DataSet to update with records from.
- **srcTable** The name of the source table to use for table mapping.

Return value

The number of rows successfully updated from the DataSet.

Remarks

For more information, see DbDataAdapter.Update.

Exceptions

- System.ArgumentNullException - The *dataSet* or *srcTable* parameter is invalid.
- InvalidOperationException - The SelectCommand property is invalid or, the table mapping is missing and MissingMappingAction property is set to MissingMappingAction.Error or, the DataTable is missing from the DataSet.

Update method

Updates the rows in the database with the changes made to the specified DataTable.

Prototypes

```
' Visual Basic

Overloads Public Function Update( _
   ByVal dataTable As DataTable _
) As Integer

// C#

public int Update(
   DataTable dataTable
);
```

Parameters

- **dataTable** A DataTable to update from.

Return value

The number of rows successfully updated from the DataTable.

Remarks

For more information, see DbDataAdapter.Update.

Exceptions

- System.ArgumentNullException - The *dataTable* parameter is invalid.
- InvalidOperationException - The SelectCommand property is invalid or, the table mapping is missing and MissingMappingAction property is set to MissingMappingAction.Error.

Update method

Updates the rows in the database with the changes made to the specified DataRow array.

Prototypes

' Visual Basic

```
Overloads Public Function Update( _
  ByVal dataRows As DataRow() _
) As Integer
```

// C#

```
public int Update(
  DataRow[] dataRows
);
```

Parameters

- **dataRows** An array of DataRow to update from.

Return value

The number of rows successfully updated from the DataRow array.

Remarks

For more information, see DbDataAdapter.Update.

Exceptions

- System.ArgumentNullException - The *dataRows* parameter is invalid.
- InvalidOperationException - The SelectCommand property is invalid or, the table mapping is missing and MissingMappingAction property is set to MissingMappingAction.Error.

FillError event

Occurs when an error is detected during a fill operation.

Prototypes

' Visual Basic

```
Public Event FillError As FillErrorEventHandler
```

// C#

```
public event FillErrorEventHandler FillError;
```

Remarks

The FillError event allows you to determine whether the fill operation should continue after the error occurs.

Examples of when the FillError event might occur are:

- The data being added to a DataSet cannot be converted to a common language runtime type without losing precision.
- The row being added contains data that violates a Constraint that must be enforced on a DataColumn in the DataSet.

To process row fill error events, you must create a FillErrorEventHandler delegate and attach it to this event.

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.

Prototypes

' Visual Basic

Public Event **RowUpdated** As ULRowUpdatedEventHandler

// C#

public event ULRowUpdatedEventHandler **RowUpdated**;

Remarks

To process row updated events, you must create a ULRowUpdatedEventHandler delegate delegate and attach it to this event.

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.

Prototypes

' Visual Basic

Public Event **RowUpdating** As ULRowUpdatingEventHandler

// C#

public event ULRowUpdatingEventHandler **RowUpdating**;

Remarks

To process row updating events, you must create a ULRowUpdatingEventHandler delegate delegate and attach it to this event.

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.

Prototypes

' Visual Basic

NotInheritable Public 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) DatabaseManager property.

To use the UltraLite Engine runtime of UltraLite.NET, set RuntimeType property to the appropriate value before using any other UltraLite.NET API.

ULDatabaseManager members

Public static properties (shared)

Member name	Description
RuntimeType property	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	Creates a new UltraLite database.
DropDatabase method	Deletes the specified database. You cannot drop a database that has open connections.
SetActiveSyncListener method	Specifies the listener object used to process ActiveSync calls from the MobiLink provider for ActiveSync.
SetServerSyncListener method	Specifies the listener object used to process the specified server synchronization message.
SignalSyncIsComplete method	Signals the MobiLink provider for ActiveSync that an application has completed synchronization.

RuntimeType property

Specifies the UltraLite.NET runtime type. The runtime type must be selected before using any other UltraLite.NET API.

Prototypes

' Visual Basic

Public Shared Property **RuntimeType** As ULRuntimeType

// C#

public const ULRuntimeType **RuntimeType** {get;set;}

Property value

A ULRuntimeType enumeration 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
```

CreateDatabase method

Creates a new UltraLite database.

Prototypes

' 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 the ULConnectionParms class.
- **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 the ULCreateParms class.

Remarks

To specify a collation, you must first include the appropriate collation data source file from the src\ulcollations\cs (for C# projects) or src\ulcollations\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.

Exceptions

- ULException class - A SQL error occurred.

Example

The following code creates the database \UltraLite\MyDatabase.udb on a Windows CE device then opens a connection to it.

```
' Visual Basic
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnCE = "\UltraLite\MyDatabase.udb"
' Assumes file src\ulcollations\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 src\ulcollations\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 161
- “ULDatabaseManager members” on page 161
- “Open method” on page 104

DropDatabase method

Deletes the specified database.

You cannot drop a database that has open connections.

Prototypes

```
' 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 the ULConnectionParms class.

Exceptions

- ULException class - A SQL error occurred.

Example

The following code creates the database \UltraLite\MyDatabase.udb on a Windows CE 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 161
- "ULDatabaseManager members" on page 161
- "Open method" on page 104

SetActiveSyncListener method

Specifies the listener object used to process ActiveSync calls from the MobiLink provider for ActiveSync.

Prototypes

' 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 interface 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 method or SetActiveSyncListener method with a different *appClassName* fail.

To remove the ActiveSync listener, call SetActiveSyncListener method with a null reference (Nothing in Visual Basic) as the *listener* parameter.

To remove all listeners, call SetServerSyncListener method with a null reference (Nothing in Visual Basic) for all parameters.

Applications should remove all listeners prior to exiting.

Exceptions

- ULException class - A SQL error occurred.

Example

Refer to the ActiveSyncInvoked method documentation for an example of SetActiveSyncListener method.

SetServerSyncListener method

Specifies the listener object used to process the specified server synchronization message.

Prototypes

```
' 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 interface 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 method or SetActiveSyncListener method with a different *appClassName* fail.

To remove the listener for a particular message, call SetServerSyncListener method with a null reference (Nothing in Visual Basic) as the *listener* parameter.

To remove all listeners, call SetServerSyncListener method with a null reference (Nothing in Visual Basic) for all parameters.

Applications should remove all listeners before exiting.

Exceptions

- ULException class - A SQL error occurred.

Example

See the ServerSyncInvoked method documentation for an example of SetServerSyncListener method.

SignalSyncIsComplete method

Signals the MobiLink provider for ActiveSync that an application has completed synchronization.

Prototypes

' Visual Basic

Public Sub **SignalSyncIsComplete()**

// C#

public void **SignalSyncIsComplete();**

Example

Refer to the ActiveSyncInvoked method documentation for an example of SignalSyncIsComplete method.

ULDatabaseSchema class

UL Ext.: Represents the schema of an UltraLite.NET database.

Prototypes

' Visual Basic

NotInheritable Public Class **ULDatabaseSchema**

// C#

public sealed class **ULDatabaseSchema**

Remarks

There is no constructor for this class. A ULDatabaseSchema class object is attached to a connection as its Schema property and is only valid while that connection is open.

ULDatabaseSchema members

Public properties

Member name	Description
CollationName property	The name of the database's collation sequence.
IsCaseSensitive property	Checks whether the database is case sensitive.
IsOpen property	Whether the database schema is open.
PublicationCount property	The number of publications in the database.
TableCount property	The number of tables in the database.

Public methods

Member name	Description
GetDatabaseProperty method	Returns the value of the specified database property.
GetPublicationName method	Returns the name of the publication identified by the specified publication ID. Publication IDs are not publication masks.
GetPublicationSchema method	Returns the publication schema corresponding to the named publication.
GetTableCountInPublications method	Returns the number of tables included in the specified publication mask.
GetTableName method	Returns the name of the table identified by the specified table ID.

Member name	Description
SetDatabaseOption method	Sets the value for the specified database option.

CollationName property

The name of the database's collation sequence.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULDatabaseSchema class” on page 168
- “ULDatabaseSchema members” on page 168
- “GetDatabaseProperty method” on page 171

IsCaseSensitive property

Checks whether the database is case sensitive.

Prototypes

' 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 UserID property and Password property are verified.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDatabaseSchema class" on page 168
- "ULDatabaseSchema members" on page 168
- "GetDatabaseProperty method" on page 171

IsOpen property

Whether the database schema is open.

Prototypes

' 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.

PublicationCount property

The number of publications in the database.

Prototypes

' 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. Publication IDs are not publication masks.

Note: Publication IDs, masks, and counts may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached IDs, masks, and counts after a schema upgrade.

See also

- "ULDatabaseSchema class" on page 168
- "ULDatabaseSchema members" on page 168
- "GetPublicationName method" on page 173

TableCount property

The number of tables in the database.

Prototypes

' 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.

GetDatabaseProperty method

Returns the value of the specified database property.

Prototypes

' 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 UserID property and Password property 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.
CollationName	The name of the database's collation sequence.
ConnCount	The number of connections to the database.
date_format	The date format used for string conversions by the database. This format is not necessarily the same as the one used by 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.
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.

Property	Description
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 TimeSpan.
timestamp_format	The timestamp format used for string conversions by the database. This format is not necessarily the same as the one used by DateTime.
timestamp_increment	The minimum difference between two unique timestamps, in nanoseconds (1,000,000th of a second).

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDatabaseSchema class" on page 168
- "ULDatabaseSchema members" on page 168
- "SetDatabaseOption method" on page 176
- "CollationName property" on page 169

GetPublicationName method

Returns the name of the publication identified by the specified publication ID. Publication IDs are not publication masks.

Prototypes

' 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 property].

Return value

The publication name as a string.

Remarks

Note: Publication IDs, masks, and counts may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached IDs, masks, and counts after a schema upgrade.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDatabaseSchema class" on page 168
- "ULDatabaseSchema members" on page 168
- "PublicationCount property" on page 170

GetPublicationSchema method

Returns the publication schema corresponding to the named publication.

Prototypes

```
' 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 class object representing the named publication.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDatabaseSchema class" on page 168
- "ULDatabaseSchema members" on page 168
- "GetPublicationName method" on page 173
- "ULPublicationSchema class" on page 286

GetTableCountInPublications method

Returns the number of tables included in the specified publication mask.

Prototypes

' Visual Basic

```
Public Function GetTableCountInPublications( _
  ByVal mask As Integer _
) As Integer
```

// C#

```
public int GetTableCountInPublications(
  int mask
);
```

Parameters

- **mask** The set of publications to check. For information on building publication masks, see ULPublicationSchema class.

Return value

The number of tables included in set of publications.

Remarks

The count will not include tables whose names end in _nosync.

Note: Publication IDs, masks, and counts may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached IDs, masks, and counts after a schema upgrade.

Exceptions

- ULException class - A SQL error occurred.

GetTableName method

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

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDatabaseSchema class" on page 168
- "ULDatabaseSchema members" on page 168
- "TableCount property" on page 171

SetDatabaseOption method

Sets the value for the specified database option.

Prototypes

```
' 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,UInt32.MaxValue]. The default is INVALID_DATABASE_ID field (used to indicate that the database ID has not been set for the current database).
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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDatabaseSchema class" on page 168
- "ULDatabaseSchema members" on page 168
- "GetDatabaseProperty method" on page 171

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.

Prototypes

' Visual Basic

Public Class **ULDataReader**
  Inherits MarshalByRefObject

// C#

public class **ULDataReader** :
  MarshalByRefobject

Remarks

There is no constructor for ULDataReader. To get a ULDataReader object, execute a ULCommand class:

```
' 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 ExecuteResultSet method, ExecuteTable method, or a ULDataAdapter class. The ULDataReader retrieves rows as needed, whereas the ULDataAdapter class 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 method.

Implements: IDataReader, IDataRecord, IDisposable

ULDataReader members

Public properties

Member name	Description
Depth property	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
FieldCount property	Returns the number of columns in the cursor.
IsBOF property	**UL Ext.:** Checks whether the current row position is before the first row.
IsClosed property	Checks whether the cursor is currently open.
IsEOF property	**UL Ext.:** Checks whether the current row position is after the last row.
Item property	Returns the value of the specified column in its native format. In C#, this property is the indexer for the ULDataReader class.
Item property	Returns the value of the specified named column in its native format. In C#, this property is the indexer for the ULDataReader class.
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.
RowCount property	**UL Ext.:** Returns the number of rows in the cursor.
Schema property	**UL Ext.:** Holds the schema of this cursor.

Public methods

Member name	Description
Close method	Closes the cursor.
Dispose method	Releases the unmanaged resources used by the ULDataReader and optionally releases the managed resources.
GetBoolean method	Returns the value for the specified column as a Boolean.
GetByte method	Returns the value for the specified column as an unsigned 8-bit value (Byte).
GetBytes method	Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination Byte array.
GetBytes method	**UL Ext.:** Returns the value for the specified column as an array of Bytes. Only valid for columns of type ULDbType enumeration, ULDbType enumeration, or ULDbType enumeration.
GetChar method	This method is not supported in UltraLite.NET.

Member name	Description
GetChars method	Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination System.Char array.
GetData method	This method is not supported in UltraLite.NET.
GetDataTypeName method	Returns the name of the specified column's provider data type.
GetDateTime method	Returns the value for the specified column as a DateTime with millisecond accuracy.
GetDecimal method	Returns the value for the specified column as a Decimal.
GetDouble method	Returns the value for the specified column as a Double.
GetFieldType method	Returns the Type most appropriate for the specified column.
GetFloat method	Returns the value for the specified column as a Single.
GetGuid method	Returns the value for the specified column as a UUID (Guid).
GetInt16 method	Returns the value for the specified column as an Int16.
GetInt32 method	Returns the value for the specified column as an Int32.
GetInt64 method	Returns the value for the specified column as an Int64.
GetName method	Returns the name of the specified column.
GetOrdinal method	Returns the column ID of the named column.
GetSchemaTable method	Returns a DataTable that describes the column metadata of the ULDataReader.
GetString method	Returns the value for the specified column as a String.
GetTimeSpan method	Returns the value for the specified column as a TimeSpan with millisecond accuracy.
GetUInt16 method	Returns the value for the specified column as a UInt16.
GetUInt32 method	Returns the value for the specified column as a UInt32.
GetUInt64 method	Returns the value for the specified column as a UInt64.
GetValue method	Returns the value of the specified column in its native format.
GetValues method	Returns all the column values for the current row.
IsDBNull method	Checks whether the value from the specified column is NULL.
MoveAfterLast method	**UL Ext.:** Positions the cursor to after the last row of the cursor.

Member name	Description
MoveBeforeFirst method	**UL Ext.:** Positions the cursor to before the first row of the cursor.
MoveFirst method	**UL Ext.:** Positions the cursor to the first row of the cursor.
MoveLast method	**UL Ext.:** Positions the cursor to the last row of the cursor.
MoveNext method	**UL Ext.:** Positions the cursor to the next row or after the last row if the cursor was already on the last row.
MovePrevious method	**UL Ext.:** Positions the cursor to the previous row or before the first row.
MoveRelative method	**UL Ext.:** Positions the cursor relative to the current row.
NextResult method	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
Read method	Positions the cursor to the next row, or after the last row if the cursor was already on the last row.

Protected methods

Member name	Description
Finalize method	Releases the unmanaged resources used by the ULDataReader.

Depth property

Returns the depth of nesting for the current row. The outermost table has a depth of zero.

Prototypes

' Visual Basic

```
NotOverridable Public Readonly Property Depth As Integer _
  Implements IDataReader.Depth
```

// C#

```
public int Depth {get;}
```

Property value

All UltraLite.NET result sets have a depth of zero.

Exceptions

- ULException class - The ULDataReader is not opened.

Implements

IDataReader.Depth

FieldCount property

Returns the number of columns in the cursor.

Prototypes

' Visual Basic

```
NotOverridable Public Readonly Property FieldCount As Integer _
  Implements IDataRecord.FieldCount
```

// C#

```
public int FieldCount {get;}
```

Return 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 ColumnCount property.

Implements

IDataRecord.FieldCount

IsBOF property

UL Ext.: Checks whether the current row position is before the first row.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

IsClosed property

Checks whether the cursor is currently open.

Prototypes

' Visual Basic

```
NotOverridable Public Readonly Property IsClosed As Boolean _
  Implements IDataReader.IsClosed
```

// C#

```
public bool IsClosed {get;}
```

Property value

True if the cursor is currently open, false if the cursor is closed.

Implements

IDataReader.IsClosed

IsEOF property

UL Ext.: Checks whether the current row position is after the last row.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

Item property

Returns the value of the specified column in its native format. In C#, this property is the indexer for the ULDataReader class.

Prototypes

' Visual Basic

```
NotOverridable Public Readonly Property Item As Object _
  Implements IDataRecord.Item
```



```
// C#

public object Item {get;}
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-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 GetValue method.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.Item

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetFieldType method" on page 198
- "Item property" on page 184

Item property

Returns the value of the specified named column in its native format. In C#, this property is the indexer for the ULDataReader class.

Prototypes

```
' Visual Basic

NotOverridable Public Readonly Property Item As Object _
  Implements IDataRecord.Item
```

```
// C#

public object Item {get;}
```

Parameters

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

Return 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 ) )
```

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.Item

See also

- “ULDataReader class” on page 178
- “ULDataReader members” on page 179
- “Item property” on page 183
- “GetOrdinal method” on page 203
- “GetValue method” on page 210
- “GetFieldType method” on page 198

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.

Prototypes

' Visual Basic

```
NotOverridable Public Readonly Property RecordsAffected As Integer _
  Implements IDataReader.RecordsAffected
```

// C#

```
public int RecordsAffected {get;}
```

Property value

The number of rows changed, inserted, or deleted by execution of the SQL statement.

Implements

IDataReader.RecordsAffected

RowCount property

UL Ext.: Returns the number of rows in the cursor.

Prototypes

' 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 StopSynchronizationDelete method.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULDataReader class” on page 178
- “ULDataReader members” on page 179
- “StartSynchronizationDelete method” on page 106
- “StopSynchronizationDelete method” on page 107

Schema property

UL Ext.: Holds the schema of this cursor.

Prototypes

' Visual Basic

Public Readonly Property **Schema** As ULCursorSchema

// C#

public ULCursorSchema **Schema** {get;}

Property value

For result sets, the ULResultSetSchema class object representing the schema of the result set. For tables, the ULTableSchema class 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 GetSchemaTable method.

Close method

Closes the cursor.

Prototypes

' Visual Basic

```
NotOverridable Public Sub Close() _
  Implements IDataReader.Close
```

// C#

```
public void Close();
```

Remarks

It is not an error to close a cursor that is already closed.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataReader.Close

Dispose method

Releases the unmanaged resources used by the ULDataReader and optionally releases the managed resources.

Prototypes

' Visual Basic

```
NotOverridable Public Sub Dispose() _
  Implements IDisposable.Dispose
```

// C#

```
public void Dispose();
```

Exceptions

- ULException class - A SQL error occurred.

Implements

IDisposable.Dispose

Finalize method

Releases the unmanaged resources used by the ULDataReader.

Prototypes

' Visual Basic

Overrides Protected Sub **Finalize()**

// C#

protected override void **Finalize();**

GetBoolean method

Returns the value for the specified column as a Boolean.

Prototypes

' Visual Basic

```
NotOverridable Public Function GetBoolean( _
  ByVal columnID As Integer _
) As Boolean _
  Implements IDataRecord.GetBoolean
```

// C#

```
public bool GetBoolean(
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a Boolean.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetBoolean

See also

- “ULDataReader class” on page 178
- “ULDataReader members” on page 179
- “GetOrdinal method” on page 203
- “GetFieldType method” on page 198

GetByte method

Returns the value for the specified column as an unsigned 8-bit value (Byte).

Prototypes

' Visual Basic

```
NotOverridable Public Function GetByte( _
   ByVal columnID As Integer _
) As Byte _
   Implements IDataRecord.GetByte
```

// C#

```
public byte GetByte(
   int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a Byte.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetByte

See also

- “ULDataReader class” on page 178
- “ULDataReader members” on page 179
- “GetOrdinal method” on page 203
- “GetFieldType method” on page 198

GetBytes method

Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination Byte array.

Prototypes

' Visual Basic

```
Overloads NotOverridable Public Function GetBytes( _
   ByVal columnID As Integer, _
   ByVal srcOffset As Long, _
   ByVal dst As Byte(), _
```

```
    ByVal dstOffset As Integer, _
    ByVal count As Integer _
) As Long _
    Implements IDataRecord.GetBytes

// C#

public 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,FieldCount property-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 class with code ULSQLCode enumeration 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 class with the appropriate error code is thrown.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetBytes

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198
- "GetBytes method" on page 191

GetBytes method

UL Ext.: Returns the value for the specified column as an array of Bytes. Only valid for columns of type ULDbType enumeration, ULDbType enumeration, or ULDbType enumeration.

Prototypes

' Visual Basic

```
Overloads 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,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as an array of Bytes.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198
- "GetBytes method" on page 189

GetChar method

This method is not supported in UltraLite.NET.

Prototypes

' Visual Basic

```
NotOverridable Public Function GetChar( _
  ByVal columnID As Integer _
) As Char _
  Implements IDataRecord.GetChar
```

// C#

```
public char GetChar(
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

This method is not supported in UltraLite.NET.

Exceptions

- ULException class - This method is not supported in UltraLite.NET.

Implements

IDataRecord.GetChar

See also

- “ULDataReader class” on page 178
- “ULDataReader members” on page 179
- “GetOrdinal method” on page 203
- “GetFieldType method” on page 198
- “GetString method” on page 206

GetChars method

Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination System.Char array.

Prototypes

' Visual Basic

```
NotOverridable Public Function GetChars( _
  ByVal columnID As Integer, _
```



```
    ByVal srcOffset As Long, _
    ByVal dst As Char(), _
    ByVal dstOffset As Integer, _
    ByVal count As Integer _
) As Long _
    Implements IDataRecord.GetChars

// C#

public 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,FieldCount property-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 class with code ULSQLCode enumeration 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 class with the appropriate error code is thrown.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetChars

See also

- “ULDataReader class” on page 178
- “ULDataReader members” on page 179
- “GetOrdinal method” on page 203
- “GetFieldType method” on page 198

GetData method

This method is not supported in UltraLite.NET.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetData( _
   ByVal i As Integer _
) As IDataReader _
   Implements IDataRecord.GetData

// C#

public IDataReader GetData(
   int i
);
```

Parameters

- **i** An integer value.

Return value

This method is not supported in UltraLite.NET.

Exceptions

- ULException class - This method is not supported in UltraLite.NET.

Implements

IDataRecord.GetData

GetDataTypeName method

Returns the name of the specified column's provider data type.

Prototypes

' Visual Basic

```
NotOverridable Public Function GetDataTypeName( _
  ByVal columnID As Integer _
) As String _
  Implements IDataRecord.GetDataTypeName
```

// C#

```
public string  GetDataTypeName(
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

A string corresponding to the column's ULDbType enumeration.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetDataTypeName

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetColumnULDbType method" on page 139

GetDateTime method

Returns the value for the specified column as a DateTime with millisecond accuracy.

Prototypes

' Visual Basic

```
NotOverridable Public Function GetDateTime( _
  ByVal columnID As Integer _
) As Date _
  Implements IDataRecord.GetDate
```

// C#

```
public DateTime GetDateTime(
```

```
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a DateTime.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetDateTime

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetDecimal method

Returns the value for the specified column as a Decimal.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetDecimal( _
  ByVal columnID As Integer _
) As Decimal _
  Implements IDataRecord.GetDecimal

// C#

public decimal GetDecimal(
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a Decimal.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetDecimal

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetDouble method

Returns the value for the specified column as a Double.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetDouble( _
  ByVal columnID As Integer _
) As Double _
  Implements IDataRecord.GetDouble

// C#

public double GetDouble(
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a Double.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetDouble

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203

- "GetFieldType method" on page 198

GetFieldType method

Returns the Type most appropriate for the specified column.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetFieldType( _
  ByVal columnID As Integer _
) As Type _
  Implements IDataRecord.GetFieldType

// C#

public Type GetFieldType(
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

A Type value for the column.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetFieldType

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetDataTypeName method" on page 194
- "GetColumnULDbType method" on page 139

GetFloat method

Returns the value for the specified column as a Single.

Prototypes

```
' Visual Basic
```



```
NotOverridable Public Function GetFloat( _
  ByVal columnID As Integer _
) As Single _
  Implements IDataRecord.GetFloat

// C#

public float  GetFloat(
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a Single.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetFloat

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetGuid method

Returns the value for the specified column as a UUID (Guid).

Prototypes

```
' Visual Basic

NotOverridable Public Function GetGuid( _
  ByVal columnID As Integer _
) As Guid _
  Implements IDataRecord.GetGuid

// C#

public Guid GetGuid(
  int columnID
);
```


Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-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 enumeration or for columns of type ULDbType enumeration with length 16.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetGuid

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198
- "GetColumnULDbType method" on page 139
- "GetColumnSize method" on page 137

GetInt16 method

Returns the value for the specified column as an Int16.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetInt16( _
   ByVal columnID As Integer _
) As Short _
   Implements IDataRecord.GetShort

// C#

public short GetInt16(
   int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as an Int16.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetInt16

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetInt32 method

Returns the value for the specified column as an Int32.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetInt32( _
   ByVal columnID As Integer _
) As Integer _
   Implements IDataRecord.GetInteger

// C#

public int GetInt32(
   int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as an Int32.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetInt32

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetInt64 method

Returns the value for the specified column as an Int64.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetInt64( _
  ByVal columnID As Integer _
) As Long _
  Implements IDataRecord.GetLong

// C#

public long GetInt64(
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as an Int64

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetInt64

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetName method

Returns the name of the specified column.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetName( _
   ByVal columnID As Integer _
) As String _
   Implements IDataRecord.GetName

// C#

public string  GetName(
   int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-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 GetColumnName method.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetName

See also

- “ULDataReader class” on page 178
- “ULDataReader members” on page 179
- “FieldCount property” on page 182
- “GetSchemaTable method” on page 204

GetOrdinal method

Returns the column ID of the named column.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetOrdinal( _
   ByVal columnName As String _
) As Integer _
   Implements IDataRecord.GetOrdinal

// C#

public 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 FieldCount property-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 GetColumnID method.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetOrdinal

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetSchemaTable method" on page 204

GetSchemaTable method

Returns a DataTable that describes the column metadata of the ULDataReader.

Prototypes

' Visual Basic

```
NotOverridable Public Function GetSchemaTable() As DataTable _
  Implements IDataReader.GetSchemaTable
```

// C#

```
public DataTable GetSchemaTable();
```

Return value

A 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 property-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 enumeration or ULDbType enumeration) or DBNull if the column is not numeric.
NumericScale	The scale of a numeric column (ProviderType ULDbType enumeration or ULDbType enumeration) or DBNull if the column is not numeric.
IsUnique	True if the column is a non-computed unique column in the table (BaseTableName) it is taken from.
IsKey	True if the column is one of a set of columns in the result set that taken together from a unique key for the result set. The set of columns with IsKey set to true does not need to be the minimal set that uniquely identifies a row in the result set.
BaseCatalogName	The name of the catalog in the database that contains the column. For UltraLite.NET, this value is always DBNull.
BaseColumnName	The original name of the column in the table BaseTableName of the database or DBNull if the column is computed or if this information cannot be determined.
BaseSchemaName	The name of the schema in the database that contains the column. For UltraLite.NET, this value is always DBNull.

DataTable column	Description
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 enumeration 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 enumeration or a ULDbType enumeration 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.

Implements

IDataReader.GetSchemaTable

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "Schema property" on page 186

GetString method

Returns the value for the specified column as a String.

Prototypes

' Visual Basic

```
NotOverridable Public Function GetString( _
  ByVal columnID As Integer _
```



```
) As String _
  Implements IDataRecord.GetString

// C#

public string  GetString(
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a String.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetString

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetTimeSpan method

Returns the value for the specified column as a TimeSpan with millisecond accuracy.

Prototypes

```
' 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,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a TimeSpan.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetUInt16 method

Returns the value for the specified column as a UInt16.

Prototypes

```
' 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,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as an UInt16.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetUInt32 method

Returns the value for the specified column as a UInt32.

Prototypes

' 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,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as an UInt32.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULDataReader class” on page 178
- “ULDataReader members” on page 179
- “GetOrdinal method” on page 203
- “GetFieldType method” on page 198

GetUInt64 method

Returns the value for the specified column as a UInt64.

Prototypes

' 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,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a UInt64

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetValue method

Returns the value of the specified column in its native format.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetValue( _
   ByVal columnID As Integer _
) As Object _
   Implements IDataRecord.GetValue

// C#

public object GetValue(
   int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-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 Item property.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetValue

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

GetValues method

Returns all the column values for the current row.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetValues( _
   ByVal values As Object() _
) As Integer _
   Implements IDataRecord.GetValues

// C#

public int GetValues(
   object[] values
);
```

Parameters

- **values** The array of 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 (FieldCount property), 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 Object array that contains fewer than the number of columns contained in the resulting row. Only the amount of data the Object array holds is copied to the array. You can also pass an 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.

Exceptions

- ULException class - A SQL error occurred.
- System.ArgumentNullException - The *values* array is NULL or has zero length.

Implements

IDataRecord.GetValues

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "FieldCount property" on page 182
- "GetFieldType method" on page 198
- "GetValue method" on page 210

IsDBNull method

Checks whether the value from the specified column is NULL.

Prototypes

```
' Visual Basic

NotOverridable Public Function IsDBNull( _
  ByVal columnID As Integer _
) As Boolean _
  Implements IDataRecord.IsDBNull

// C#

public bool IsDBNull(
  int columnID
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-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.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.IsDBNull

See also

- "ULDataReader class" on page 178
- "ULDataReader members" on page 179
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

MoveAfterLast method

UL Ext.: Positions the cursor to after the last row of the cursor.

Prototypes

' Visual Basic

Public Sub **MoveAfterLast()**

// C#

public void **MoveAfterLast();**

Exceptions

- ULException class - A SQL error occurred.

MoveBeforeFirst method

UL Ext.: Positions the cursor to before the first row of the cursor.

Prototypes

' Visual Basic

Public Sub **MoveBeforeFirst()**

// C#

public void **MoveBeforeFirst();**

Exceptions

- ULException class - A SQL error occurred.

MoveFirst method

UL Ext.: Positions the cursor to the first row of the cursor.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

MoveLast method

UL Ext.: Positions the cursor to the last row of the cursor.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

MoveNext method

UL Ext.: Positions the cursor to the next row or after the last row if the cursor was already on the last row.

Prototypes

' 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 Read method.

Exceptions

- ULException class - A SQL error occurred.

MovePrevious method

UL Ext.: Positions the cursor to the previous row or before the first row.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

MoveRelative method

UL Ext.: Positions the cursor relative to the current row.

Prototypes

' 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 (IsEOF property) if *offset* is positive, and before the first row (IsBOF property) if the *offset* is negative.

Exceptions

- ULException class - A SQL error occurred.

NextResult method

Advances the ULDataReader to the next result when reading the results of batch SQL statements.

Prototypes

' Visual Basic

```
NotOverridable Public Function NextResult() As Boolean _
  Implements IDataReader.NextResult
```

// C#

```
public 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, hence the ULDataReader is always positioned on the first and only result set. Calling NextResult has no effect.

Exceptions

- ULException class - The ULDataReader is not opened.

Implements

IDataReader.NextResult

Read method

Positions the cursor to the next row, or after the last row if the cursor was already on the last row.

Prototypes

' Visual Basic

```
NotOverridable Public Function Read() As Boolean _
  Implements IDataReader.Read
```

// C#

```
public 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 MoveNext method.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataReader.Read

ULDateOrder enumeration

UL Ext.: Enumerates the date orders that a database can support.

Prototypes

' Visual Basic

Public Enum **ULDateOrder**

// C#

public enum **ULDateOrder**

Member name

Member name	Description
DMY	Day of the month followed by month, followed by year.
MDY	Month followed by day of the month, followed by year.
YMD	Year followed by month, followed by day of the month.

See also

- "DateOrder property" on page 125

ULDbType enumeration

Enumerates the UltraLite.NET database data types.

Prototypes

' 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.
UnsignedSmall-Int	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.

Member name

Member name	Description
BigInt	Signed 64-bit integer.
Binary	Binary data, with a specified maximum length. The enumeration values **Binary** and **VarBinary** are aliases of each other.
Bit	1-bit flag.
Char	Character data, with a specified length. In UltraLite.NET, this type always supports Unicode characters. The types **Char** and **VarChar** are fully compatible.
Date	Date information.
DateTime	Timestamp information (date, time). The enumeration values **DateTime** and **TimeStamp** are aliases of each other.
Decimal	Exact numerical data, with a specified precision and scale. The enumeration values **Decimal** and **Numeric** are aliases of each other.
Double	Double precision floating-point number (8 bytes).
Float	Single precision floating-point number (4 bytes). The enumeration values **Float** and **Real** are aliases of each other.
Integer	Unsigned 32-bit integer.

Member name	Description
LongBinary	Binary data, with variable length.
LongVarchar	Character data, with variable length. In UltraLite.NET, this type always supports Unicode characters.
Numeric	Exact numerical data, with a specified precision and scale. The enumeration values **Decimal** and **Numeric** are aliases of each other.
Real	Single precision floating-point number (4 bytes). The enumeration values **Float** and **Real** are aliases of each other.
SmallInt	Signed 16-bit integer.
Time	Time information.
TimeStamp	Timestamp information (date, time). The enumeration values **DateTime** and **TimeStamp** are aliases of each other.
TinyInt	Unsigned 8-bit integer.
UniqueIdentifier	Universally Unique Identifier (UUID/GUID).
UnsignedBigInt	Unsigned 64-bit integer.
UnsignedInt	Unsigned 32-bit integer.
UnsignedSmallInt	Unsigned 16-bit integer.
VarBinary	Binary data, with a specified maximum length. The enumeration values **Binary** and **VarBinary** are aliases of each other.
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.

See also

- "GetFieldType method" on page 198
- "GetDataTypeName method" on page 194
- "GetColumnULDbType method" on page 139

ULException class

Represents a SQL error returned by the UltraLite.NET database.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULException
  Inherits ApplicationException
```

// C#

```
public sealed class ULException :
  ApplicationException
```

Remarks

The SQL code denoting the error is returned in the NativeError property.

This class is not serializable under the .NET Compact Framework.

ULException members

Public constructors

Member name	Description
ULException constructor	Creates a ULException with the given error code.

Public properties

Member name	Description
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the System.Exception instance that caused the current exception.
Message (inherited from Exception)	Gets a message that describes the current exception.
NativeError property	Returns the SQL code returned by the database.
Source property	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 System.Exception that is the root cause of one or more subsequent exceptions.
GetObjectData method	Populates a SerializationInfo with the data needed to serialize this ULException.
ToString (inherited from Exception)	Creates and returns a string representation of the current exception.

ULException constructor

Creates a ULException with the given error code.

Prototypes

' Visual Basic

```
Overloads 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 ULSQLCode enumeration is retrieved from the **iAnywhere.Data.UltraLite.resources** assembly. Resources are searched for, by culture, using the following order: System.Globalization.CultureInfo.CurrentUICulture, then System.Globalization.CultureInfo.CurrentCulture, and finally culture "EN".

NativeError property

Returns the SQL code returned by the database.

Prototypes

' Visual Basic

Public Readonly Property **NativeError** As ULSQLCode

// C#

public ULSQLCode **NativeError** {get;}

Property value

The ULSQLCode enumeration value returned by the database.

Source property

Returns the name of the provider that generated the error.

Prototypes

' Visual Basic

Public Readonly Property **Source** As String

// C#

public string **Source** {get;}

Property value

The string value identifying UltraLite.NET as the provider.

GetObjectData method

Populates a SerializationInfo with the data needed to serialize this ULException.

Prototypes

' Visual Basic

```
Overrides Public Sub GetObjectData( _
  ByVal info As System.Runtime.Serialization.SerializationInfo, _
  ByVal context As System.Runtime.Serialization.StreamingContext _
) _
  Implements ISerializable.GetObjectData
```

// C#

```
public override void GetObjectData(
  System.Runtime.Serialization.SerializationInfo info,
```



```
    System.Runtime.Serialization.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.

Implements

ISerializable.GetObjectData

ULFileTransfer class

UL Ext.: Transfers a file from a remote database using the MobiLink server.

Prototypes

' Visual Basic

NotInheritable Public 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 RuntimeType property to the appropriate value before using this API or any other UltraLite.NET API.

To transfer a file you must set the FileName property, Stream property, UserName property, and Version property.

ULFileTransfer members

Public constructors

Member name	Description
ULFileTransfer constructor	Initializes a ULFileTransfer object. The connection must be opened before you can perform any operations against the database.

Public properties

Member name	Description
AuthenticationParms property	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
AuthStatus property	Returns the authorization status code for the last file transfer attempt.
AuthValue property	Returns the return value from custom user authentication synchronization scripts.
DestinationFileName property	Specifies the local file name for the downloaded file.
DestinationPath property	Specifies where to download the file.
DownloadedFile property	Checks whether the file was actually downloaded during the last file transfer attempt.

Member name	Description
FileAuthCode property	Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.
FileName property	Specifies the name of the file to download.
ForceDownload property	Specifies whether to force the download of the file if it exists.
Password property	The MobiLink password for the user specified by UserName.
ResumePartialDownload property	Specifies whether to resume or discard a previous partial download.
Stream property	Specifies the MobiLink synchronization stream to use for the file transfer.
StreamErrorCode property	Returns the error reported by the stream itself for the last file transfer attempt.
StreamErrorSystem property	Returns the stream error system-specific code.
StreamParms property	Specifies the parameters to configure the synchronization stream.
UserName property	The user name that uniquely identifies the MobiLink client to the MobiLink server.
Version property	Specifies which synchronization script to use.

Public methods

Member name	Description
DownloadFile method	Download the file specified by the properties of this object.
DownloadFile method	Download the file specified by the properties of this object with progress events posted to the specified listener.

ULFileTransfer constructor

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

Prototypes

' 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 RuntimeType property to the appropriate value before using this API or any other UltraLite.NET API.

The ULFileTransfer object needs to have the FileName property, Stream property, UserName property, and Version property set before it can transfer a file.

See also

- "ULFileTransfer class" on page 226
- "ULFileTransfer members" on page 226
- "DownloadFile method" on page 236

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Prototypes

' 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).

AuthStatus property

Returns the authorization status code for the last file transfer attempt.

Prototypes

' Visual Basic

Public Readonly Property **AuthStatus** As ULAuthStatusCode

// C#

public ULAuthStatusCode **AuthStatus** {get;}

Property value

One of the ULAuthStatusCode enumeration values denoting the authorization status for the last file transfer attempt.

AuthValue property

Returns the return value from custom user authentication synchronization scripts.

Prototypes

' 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.

DestinationFileName property

Specifies the local file name for the downloaded file.

Prototypes

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

DestinationPath property

Specifies where to download the file.

Prototypes

' 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 226
- "ULFileTransfer members" on page 226
- "ForceDownload property" on page 231
- "DownloadFile method" on page 236

DownloadedFile property

Checks whether the file was actually downloaded during the last file transfer attempt.

Prototypes

' 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 method is invoked, it will return true, but DownloadedFile will be false. If an error occurs and DownloadFile method returns false, DownloadedFile will be false.

FileAuthCode property

Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.

Prototypes

' 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.

FileName property

Specifies the name of the file to download.

Prototypes

' 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 226
- "ULFileTransfer members" on page 226
- "DestinationFileName property" on page 229
- "DownloadFile method" on page 236

ForceDownload property

Specifies whether to force the download of the file if it exists.

Prototypes

' 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 226
- "ULFileTransfer members" on page 226
- "ResumePartialDownload property" on page 233
- "DownloadFile method" on page 236

Password property

The MobiLink password for the user specified by UserName.

Prototypes

' 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 226

- "ULFileTransfer members" on page 226
- "UserName property" on page 235
- "DownloadFile method" on page 236

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Prototypes

' 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 226
- "ULFileTransfer members" on page 226
- "ForceDownload property" on page 231
- "DownloadFile method" on page 236

Stream property

Specifies the MobiLink synchronization stream to use for the file transfer.

Prototypes

' Visual Basic

Public Property **Stream** As ULStreamType

// C#

public ULStreamType **Stream** {get;set;}

Property value

One of the ULStreamType enumeration values specifying the type of synchronization stream to use. The default is ULStreamType enumeration.

Remarks

Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. These parameters are supplied by the StreamParms property.

If the stream type is set to a value that is invalid for the platform, the stream type is set to ULStreamType enumeration.

See also

- "ULFileTransfer class" on page 226
- "ULFileTransfer members" on page 226
- "ULStreamType enumeration" on page 354
- "StreamParms property" on page 235
- "DownloadFile method" on page 236

StreamErrorCode property

Returns the error reported by the stream itself for the last file transfer attempt.

Prototypes

' Visual Basic

Public Readonly Property **StreamErrorCode** As ULStreamErrorCode

// C#

public ULStreamErrorCode **StreamErrorCode** {get;}

Property value

One of the ULStreamErrorCode enumeration values denoting the error reported by the stream itself, ULStreamErrorCode enumeration if no error occurred.

StreamErrorSystem property

Returns the stream error system-specific code.

Prototypes

' Visual Basic

Public Readonly Property **StreamErrorSystem** As Integer

// C#

public int **StreamErrorSystem** {get;}

Property value

An integer denoting the stream error system-specific code.

StreamParms property

Specifies the parameters to configure the synchronization stream.

Prototypes

' 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 on configuring specific stream types, see "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*].

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 226
- "ULFileTransfer members" on page 226
- "Stream property" on page 233
- "DownloadFile method" on page 236
- "ULStreamType enumeration" on page 354

UserName property

The user name that uniquely identifies the MobiLink client to the MobiLink server.

Prototypes

' 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 226
- "ULFileTransfer members" on page 226
- "Password property" on page 232
- "DownloadFile method" on page 236

Version property

Specifies which synchronization script to use.

Prototypes

' 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 226
- "ULFileTransfer members" on page 226
- "DownloadFile method" on page 236

DownloadFile method

Download the file specified by the properties of this object.

Prototypes

' Visual Basic

Overloads Public Function **DownloadFile()** As Boolean

// C#

public bool **DownloadFile();**

Return value

True if successful, false otherwise (check StreamErrorCode property and other status properties for reason).

Remarks

The file specified by the FileName property is downloaded by the MobiLink server to the DestinationPath property using the Stream property, UserName property, Password property, and Version property. Other properties that affect the download are DestinationFileName property, AuthenticationParms property, ForceDownload property and ResumePartialDownload property.

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 AuthStatus property, AuthValue property, FileAuthCode property, DownloadedFile property, StreamErrorCode property, and StreamErrorSystem property.

See also

- "ULFileTransfer class" on page 226
- "ULFileTransfer members" on page 226
- "DownloadFile method" on page 237

DownloadFile method

Download the file specified by the properties of this object with progress events posted to the specified listener.

Prototypes

' Visual Basic

```
Overloads 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 StreamErrorCode property and other status properties for reason).

Remarks

The file specified by the FileName property is downloaded by the MobiLink server to the DestinationPath property using the Stream property, UserName property, Password property, and Version property. Other properties that affect the download are DestinationFileName property, AuthenticationParms property, ForceDownload property and ResumePartialDownload property.

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 AuthStatus property, AuthValue property, FileAuthCode property, DownloadedFile propertyStreamErrorCode property, and StreamErrorSystem property.

Errors may result in no data being sent to the listener.

See also

- "ULFileTransfer class" on page 226
- "ULFileTransfer members" on page 226
- "DownloadFile method" on page 236
- "ULFileTransferProgressListener interface" on page 242

ULFileTransferProgressData class

UL Ext.: Returns file transfer progress monitoring data.

Prototypes

' Visual Basic

Public Class **ULFileTransferProgressData**

// C#

public class **ULFileTransferProgressData**

See also

- "ULFileTransferProgressData members" on page 239
- "ULFileTransferProgressListener interface" on page 242

ULFileTransferProgressData members

Public static fields (shared)

Member name	Description
FLAG_IS_BLOCKING field	A flag indicating that the file transfer is blocked awaiting a response from the MobiLink server.

Public properties

Member name	Description
BytesReceived property	Returns the number of bytes received so far.
FileSize property	Returns the size of the file being transferred.
Flags property	Returns the current file transfer flags indicating additional information relating to the current state.
ResumedAtSize property	Returns the point in the file where the transfer was resumed.

FLAG_IS_BLOCKING field

A flag indicating that the file transfer is blocked awaiting a response from the MobiLink server.

Prototypes

' Visual Basic

Public Shared **FLAG_IS_BLOCKING** As Integer

// C#

public const int **FLAG_IS_BLOCKING**;

BytesReceived property

Returns the number of bytes received so far.

Prototypes

' Visual Basic

Public Readonly Property **BytesReceived** As UInt64

// C#

public ulong **BytesReceived** {get;}

Property value

The number of bytes received so far.

FileSize property

Returns the size of the file being transferred.

Prototypes

' Visual Basic

Public Readonly Property **FileSize** As UInt64

// C#

public ulong **FileSize** {get;}

Property value

The size of the file in bytes.

Flags property

Returns the current file transfer flags indicating additional information relating to the current state.

Prototypes

' 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 239
- “ULFileTransferProgressData members” on page 239
- “FLAG_IS_BLOCKING field” on page 239

ResumedAtSize property

Returns the point in the file where the transfer was resumed.

Prototypes

' Visual Basic

Public Readonly Property **ResumedAtSize** As UInt64

// C#

public ulong **ResumedAtSize** {get;}

Property value

The number of bytes transferred previously.

ULFileTransferProgressListener interface

UL Ext.: The listener interface for receiving file transfer progress events.

Prototypes

' Visual Basic

Public Interface **ULFileTransferProgressListener**

// C#

public interface **ULFileTransferProgressListener**

See also

- "ULFileTransferProgressListener members" on page 242
- "DownloadFile method" on page 237

ULFileTransferProgressListener members

Public methods

Member name	Description
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.

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.

Prototypes

' Visual Basic

```
Public Function FileTransferProgressed( _
  ByVal data As ULFileTransferProgressData _
) As Boolean
```

// C#

```
public bool FileTransferProgressed(
  ULFileTransferProgressData data
);
```

Parameters

- **data** A ULFileTransferProgressData class 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.

ULIndexSchema class

UL Ext.: Represents the schema of an UltraLite table index.

Prototypes

' Visual Basic

NotInheritable Public Class **ULIndexSchema**

// C#

public sealed class **ULIndexSchema**

Remarks

There is no constructor for this class. Index schemas are created using the PrimaryKey property, GetIndex method, and GetOptimalIndex method of the ULTableSchema class.

ULIndexSchema members

Public properties

Member name	Description
ColumnCount property	Returns the number of columns in the index.
IsForeignKey property	Checks whether the index is a foreign key.
IsForeignKeyCheckOnCommit property	Checks whether referential integrity for the foreign key is performed on commits or on inserts and updates.
IsForeignKeyNullable property	Checks whether the foreign key is nullable.
IsOpen property	Determines whether the index schema is open or closed.
IsPrimaryKey property	Checks whether the index is the primary key.
IsUniqueIndex property	Checks whether the index is unique.
IsUniqueKey property	Checks whether the index is a unique key.
Name property	Returns the name of the index.
ReferencedIndexName property	The name of the referenced primary index if the index is a foreign key.
ReferencedTableName property	The name of the referenced primary table if the index is a foreign key.

Public methods

Member name	Description
GetColumnName method	Returns the name of the *colOrdinalInIndex*'th column in this index.
IsColumnDescending method	Checks whether the named column is used in descending order by the index.

ColumnCount property

Returns the number of columns in the index.

Prototypes

' 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.

IsForeignKey property

Checks whether the index is a foreign key.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

IsForeignKeyCheckOnCommit property

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

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred (including index is not a foreign key).

See also

- “ULIndexSchema class” on page 244
- “ULIndexSchema members” on page 244
- “IsForeignKey property” on page 245

IsForeignKeyNullable property

Checks whether the foreign key is nullable.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred (including index is not a foreign key).

See also

- "ULIndexSchema class" on page 244
- "ULIndexSchema members" on page 244
- "IsForeignKey property" on page 245

IsOpen property

Determines whether the index schema is open or closed.

Prototypes

' Visual Basic

Public Readonly Property **IsOpen** As Boolean

// C#

public bool **IsOpen** {get;}

Property value

True if the index schema is open, otherwise false.

IsPrimaryKey property

Checks whether the index is the primary key.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

IsUniqueIndex property

Checks whether the index is unique.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

IsUniqueKey property

Checks whether the index is a unique key.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

Name property

Returns the name of the index.

Prototypes

' Visual Basic

Public Readonly Property **Name** As String

// C#

public string **Name** {get;}

Property value

A string specifying the name of the index.

Exceptions

- ULException class - A SQL error occurred.

ReferencedIndexName property

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

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred (including index is not a foreign key).

See also

- “ULIndexSchema class” on page 244
- “ULIndexSchema members” on page 244
- “IsForeignKey property” on page 245

ReferencedTableName property

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

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred (including index is not a foreign key).

See also

- "ULIndexSchema class" on page 244
- "ULIndexSchema members" on page 244
- "IsForeignKey property" on page 245

GetColumnName method

Returns the name of the *colOrdinalInIndex*'th column in this index.

Prototypes

```
' 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 property].

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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULIndexSchema class" on page 244
- "ULIndexSchema members" on page 244
- "ColumnCount property" on page 245

IsColumnDescending method

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

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULIndexSchema class" on page 244
- "ULIndexSchema members" on page 244
- "GetColumnName method" on page 250
- "ColumnCount property" on page 245

ULInfoMessageEventArgs class

Provides data for the InfoMessage event event.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULInfoMessageEventArgs
  Inherits EventArgs
```

// C#

```
public sealed class ULInfoMessageEventArgs :
  EventArgs
```

ULInfoMessageEventArgs members

Public properties

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

Public methods

Member name	Description
ToString method	Returns the string representation of the InfoMessage event event.

Message property

The informational or warning message string returned by the database.

Prototypes

' Visual Basic

```
Public Readonly Property Message As String
```

// C#

```
public string  Message {get;}
```

Property value

A string containing the informational or warning message.

NativeError property

The SQL code corresponding to the informational message or warning returned by the database.

Prototypes

' Visual Basic

Public Readonly Property **NativeError** As ULSQLCode

// C#

public ULSQLCode **NativeError** {get;}

Property value

An informational or warning ULSQLCode enumeration value.

Source property

The name of the ADO.NET data provider returning the message.

Prototypes

' Visual Basic

Public Readonly Property **Source** As String

// C#

public string **Source** {get;}

Property value

The string "UltraLite.NET Data Provider".

ToString method

Returns the string representation of the InfoMessage event event.

Prototypes

' Visual Basic

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

// C#

public override string **ToString();**

Property value

The string "UltraLite.NET Data Provider".

Return value

The informational or warning message string.

ULInfoMessageEventHandler delegate

Represents the method that will handle the InfoMessage event event.

Prototypes

```
' Visual Basic

Public Delegate Sub ULInfoMessageEventHandler( _
  ByVal obj As Object, _
  ByVal args As ULInfoMessageEventArgs _
)

// C#

public delegate void ULInfoMessageEventHandler(
  object obj,
  ULInfoMessageEventArgs args
);
```

Parameters

- **obj** The connection sending the event.
- **args** The ULInfoMessageEventArgs class object that contains the event data.

ULParameter class

Represents a parameter to a ULCommand class.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULParameter
  Inherits MarshalByRefObject
```

// C#

```
public sealed class ULParameter :
  MarshalByRefobject
```

Remarks

A ULParameter object can be created directly using one of its many constructors, or using the 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 constructor 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 CreateParameter method) must be added to a Parameters property 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 Value property is important.

Implements: IDbDataParameter, IDataParameter

ULParameter members

Public constructors

Member name	Description
ULParameter constructor	Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.
ULParameter constructor	Initializes a ULParameter object with the specified parameter name and value.
ULParameter 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.

Member name	Description
ULParameter 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.
ULParameter 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.
ULParameter 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.

Public properties

Member name	Description
DbType property	Specifies the DbType of the parameter
Direction property	A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.
IsNullable property	Specifies whether the parameter accepts null values.
Offset property	Specifies the offset to the Value property.
ParameterName property	Specifies the name of the parameter.
Precision property	Specifies the maximum number of digits used to represent the Value property.
Scale property	Specifies the number of decimal places to which Value property is resolved.
Size property	Specifies the maximum size of the data within the column.
SourceColumn property	Specifies the name of the source column mapped to the DataSet and used for loading or returning the value.
SourceVersion property	The DataRowVersion to use when loading Value property.
ULDbType property	Specifies the ULDbType enumeration of the parameter
Value property	Specifies the value of the parameter.

Public methods

Member name	Description
ToString method	Returns the string representation of this instance.

ULParameter constructor

Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.

Prototypes

' Visual Basic

Overloads Public Sub **New()**

// C#

public **ULParameter();**

Example

The following code creates a ULParameter with the value 3 and adds it to a ULCommand class 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 256
- "ULParameter members" on page 256
- "Value property" on page 269
- "ULParameter constructor" on page 258

ULParameter constructor

Initializes a ULParameter object with the specified parameter name and value.

Prototypes

' Visual Basic

Overloads 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 class.
- **value** An 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 class 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 256
- "ULParameter members" on page 256
- "ULParameter constructor" on page 258

ULParameter 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.

Prototypes

' Visual Basic

```
Overloads 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 class.

- **dbType** One of the ULDbType enumeration values.

Remarks

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property is important.

See also

- "ULParameter class" on page 256
- "ULParameter members" on page 256
- "ULParameter constructor" on page 258
- "ULParameter constructor" on page 258

ULParameter 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.

Prototypes

```
' Visual Basic

Overloads 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 class.
- **dbType** One of the ULDbType enumeration 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 Value property is important.

See also

- "ULParameter class" on page 256

- "ULParameter members" on page 256
- "ULParameter constructor" on page 258
- "ULParameter constructor" on page 258

ULParameter 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.

Prototypes

```
' Visual Basic

Overloads 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 class.
- **dbType** One of the ULDbType enumeration 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 Value property is important.

See also

- "ULParameter class" on page 256
- "ULParameter members" on page 256
- "ULParameter constructor" on page 258
- "ULParameter constructor" on page 258

ULParameter 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.

Prototypes

' Visual Basic

```
Overloads 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 class.
- **dbType** One of the ULDbType enumeration values.
- **size** The length of the parameter.
- **direction** One of the 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 DataRowVersion values.
- **value** An Object that is to be the value of the parameter.

Exceptions

- ULException class - Only the ParameterDirection.Input direction is supported in UltraLite.NET.

See also

- "ULParameter class" on page 256
- "ULParameter members" on page 256
- "ULParameter constructor" on page 258
- "ULParameter constructor" on page 258

DbType property

Specifies the DbType of the parameter

Prototypes

' Visual Basic

```
NotOverridable Public Property DbType As DbType _
  Implements IDataParameter.DbType
```

// C#

```
public DbType DbType {get;set;}
```

Property value

One of the DbType values.

Remarks

The ULDbType property and DbType properties are linked. Therefore, setting the DbType changes the ULDbType property to a supporting ULDbType enumeration.

Exceptions

- ArgumentException - There is no mapping from the specified value to a ULDbType enumeration, hence, the specified value is not supported.

Implements

IDataParameter.DbType

Direction property

A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

Prototypes

' Visual Basic

```
NotOverridable Public Property Direction As ParameterDirection _
  Implements IDataParameter.Direction
```

// C#

```
public ParameterDirection Direction {get;set;}
```

Property value

One of the ParameterDirection values.

Remarks

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property is important.

Exceptions

- ULException class - Only the ParameterDirection.Input direction is supported in UltraLite.NET.

Implements

IDataParameter.Direction

IsNullable property

Specifies whether the parameter accepts null values.

Prototypes

' Visual Basic

```
Public Property IsNullable As Boolean _
  Implements IDataParameter.IsNullable
```

// C#

```
public 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 Value property is important.

Implements

IDataParameter.IsNullable

Offset property

Specifies the offset to the Value property.

Prototypes

' 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 Value property is important.

ParameterName property

Specifies the name of the parameter.

Prototypes

' Visual Basic

```
NotOverridable Public Property ParameterName As String _
  Implements IDataParameter.ParameterName
```

// C#

```
public 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 class. All parameters are treated as positional parameters and are used by a command in the order that they were added.

Implements

IDataParameter.ParameterName

Precision property

Specifies the maximum number of digits used to represent the Value property.

Prototypes

' Visual Basic

```
NotOverridable Public Property Precision As Byte _
  Implements IDbDataParameter.Precision
```

// C#

```
public byte Precision {get;set;}
```

Property value

The maximum number of digits used to represent the Value property. The default value is 0, which indicates that the data provider sets the precision for the Value property.

Remarks

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property is important.

Exceptions

- ArgumentException - The value is greater then 38.

Implements

IDbDataParameter.Precision

Scale property

Specifies the number of decimal places to which Value property is resolved.

Prototypes

' Visual Basic

```
NotOverridable Public Property Scale As Byte _
  Implements IDbDataParameter.Scale
```

// C#

```
public byte Scale {get;set;}
```

Property value

The number of decimal places to which Value property 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 Value property is important.

Implements

IDbDataParameter.Scale

Size property

Specifies the maximum size of the data within the column.

Prototypes

' Visual Basic

```
NotOverridable Public Property Size As Integer _
  Implements IDbDataParameter.Size
```

// C#

```
public 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 Value property is important.

Implements

IDbDataParameter.Size

SourceColumn property

Specifies the name of the source column mapped to the DataSet and used for loading or returning the value.

Prototypes

' Visual Basic

```
NotOverridable Public Property SourceColumn As String _
  Implements IDataParameter.SourceColumn
```

// C#

```
public 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 Value property is important.

Implements

IDataParameter.SourceColumn

SourceVersion property

The DataRowVersion to use when loading Value property.

Prototypes

' Visual Basic

```
NotOverridable Public Property SourceVersion As DataRowVersion _
  Implements IDataParameter.SourceVersion
```

// C#

```
public DataRowVersion SourceVersion {get;set;}
```

Implements

IDataParameter.SourceVersion

ULDbType property

Specifies the ULDbType enumeration of the parameter

Prototypes

' Visual Basic

```
Public Property ULDbType As ULDbType
```

// C#

```
public ULDbType ULDbType {get;set;}
```

Property value

One of the ULDbType enumeration values.

Remarks

The ULDbType and DbType property are linked. Therefore, setting the ULDbType changes the DbType property to a supporting DbType.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property is important.

Value property

Specifies the value of the parameter.

Prototypes

' Visual Basic

```
NotOverridable Public Property Value As Object _
  Implements IDataParameter.Value
```

// C#

```
public object Value {get;set;}
```

Property value

An 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 class with ULSQLCode enumeration if it cannot convert the value.

Implements

IDataParameter.Value

ToString method

Returns the string representation of this instance.

Prototypes

' Visual Basic

```
Overrides Public Function ToString() As String
```

// C#

```
public override string  ToString();
```

Return value

The name of the parameter.

ULParameterCollection class

Represents all parameters to a ULCommand class.

Prototypes

' Visual Basic

NotInheritable Public Class **ULParameterCollection**

// C#

public sealed class **ULParameterCollection**

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 CommandText property. 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 CommandText property 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 Parameters property.

Implements: IDataParameterCollection

ULParameterCollection members

Public properties

Member name	Description
Count property	Returns the number of ULParameter class objects in the collection.
Item property	Returns the ULParameter class at the specified index. In C#, this property is the indexer for the ULParameterCollection class.
Item property	Returns the ULParameter class with the specified name. In C#, this property is the indexer for the ULParameterCollection class.

Public methods

Member name	Description
Add method	Adds a ULParameter class to the collection.
Add method	Adds a ULParameter class to the collection.
Add method	Adds a new ULParameter class, created using the specified parameter name and value, to the collection.

Member name	Description
Add method	Adds a new ULParameter class, created using the specified parameter name and data type, to the collection.
Add method	Adds a new ULParameter class, created using the specified parameter name, data type, and length, to the collection.
Add method	Adds a new ULParameter class, created using the specified parameter name, data type, length, and source column name, to the collection.
Clear method	Removes all the parameters from the collection.
Contains method	Checks whether a ULParameter class exists in the collection.
Contains method	Checks whether a ULParameter class with the specified name exists in the collection.
CopyTo method	Copies ULParameter class objects from the ULParameterCollection to the specified array.
GetEnumerator method	Returns an enumerator for the collection.
IndexOf method	Returns the location of the ULParameter class in the collection.
IndexOf method	Returns the location of the ULParameter class with the specified name in the collection.
Insert method	Inserts an ULParameter class in the collection at the specified index.
Remove method	Removes an ULParameter class from the collection.
RemoveAt method	Removes the parameter at the specified index in the collection.
RemoveAt method	Removes the parameter with the specified name from the collection.

Count property

Returns the number of ULParameter class objects in the collection.

Prototypes

' Visual Basic

```
NotOverridable Public Readonly Property Count As Integer _
  Implements ICollection.Count
```

// C#

```
public int Count {get;}
```

Property value

The number of ULParameter class objects in the collection.

Implements

ICollection.Count

Item property

Returns the ULParameter class at the specified index. In C#, this property is the indexer for the ULParameterCollection class.

Prototypes

' Visual Basic

Public Property **Item** As ULParameter

// C#

public ULParameter **Item** {get;set;}

Parameters

- **index** The zero-based index of the parameter to retrieve. The value must be in the range [0,Count property-1]. The first parameter in the collection has an index value of zero.

Return value

The ULParameter class at the specified index.

Exceptions

- System.IndexOutOfRangeException - The index is invalid.

Item property

Returns the ULParameter class with the specified name. In C#, this property is the indexer for the ULParameterCollection class.

Prototypes

' Visual Basic

Public Property **Item** As ULParameter

// C#

public ULParameter **Item** {get;set;}

Parameters

- **parameterName** The name of the parameter to retrieve.

Return value

The ULParameter class with the specified name.

Exceptions

- System.ArgumentNullException - You cannot set a parameter using a null (Nothing in Visual Basic) parameter name.
- System.IndexOutOfRangeException - There is no parameter with the specified name.

See also

- "ULParameterCollection class" on page 270
- "ULParameterCollection members" on page 270
- "Item property" on page 183
- "GetOrdinal method" on page 203
- "GetValue method" on page 210
- "GetFieldType method" on page 198

Add method

Adds a ULParameter class to the collection.

Prototypes

' Visual Basic

```
Overloads NotOverridable Public Function Add( _
  ByVal value As Object _
) As Integer _
  Implements IList.Add
```

// C#

```
public int Add(
  object value
);
```

Parameters

- **value** The ULParameter class object to add to the collection.

Return value

The index of the new ULParameter class 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 CommandText property. 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 CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

Exceptions

- ArgumentException - The ULParameter class object can only be added to the collection once.

- System.ArgumentNullException - The value cannot be null (Nothing in Visual Basic).
- System.InvalidCastException - The value specified must be a ULParameter class.

Implements

IList.Add

See also

- "ULParameterCollection class" on page 270
- "ULParameterCollection members" on page 270
- "Add method" on page 274
- "Add method" on page 275

Add method

Adds a ULParameter class to the collection.

Prototypes

```
' Visual Basic

Overloads Public Function Add( _
  ByVal value As ULParameter _
) As ULParameter

// C#

public ULParameter Add(
  ULParameter value
);
```

Parameters

- **value** The ULParameter class object to add to the collection.

Return value

The new ULParameter class 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 CommandText property. 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 CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

Exceptions

- ArgumentException - The ULParameter class object can only be added to the collection once.
- System.ArgumentNullException - The value cannot be null (Nothing in Visual Basic).

See also

- "ULParameterCollection class" on page 270
- "ULParameterCollection members" on page 270
- "Add method" on page 275

Add method

Adds a new ULParameter class, created using the specified parameter name and value, to the collection.

Prototypes

' Visual Basic

```
Overloads 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 class.
- **value** An Object that is to be the value of the parameter.

Return value

The new ULParameter class 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 CommandText property. 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 CommandText property 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 class called cmd.

```
' Visual Basic
cmd.Parameters.Add( "", CType( 0, Object ) )
```



```
// C#
cmd.Parameters.Add( "", (object)0 );
```

See also

- "ULParameterCollection class" on page 270
- "ULParameterCollection members" on page 270
- "Add method" on page 274

Add method

Adds a new ULParameter class, created using the specified parameter name and data type, to the collection.

Prototypes

' Visual Basic

```
Overloads 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 class.
- **ulDbType** One of the ULDbType enumeration values.

Return value

The new ULParameter class 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 CommandText property. 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 CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "ULParameterCollection class" on page 270
- "ULParameterCollection members" on page 270

- "Add method" on page 274
- "Add method" on page 275

Add method

Adds a new ULParameter class, created using the specified parameter name, data type, and length, to the collection.

Prototypes

```
' Visual Basic

Overloads 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 class.
- **ulDbType** One of the ULDbType enumeration values.
- **size** The length of the parameter.

Return value

The new ULParameter class 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 CommandText property. 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 CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "ULParameterCollection class" on page 270
- "ULParameterCollection members" on page 270
- "Add method" on page 274

- "Add method" on page 275

Add method

Adds a new ULParameter class, created using the specified parameter name, data type, length, and source column name, to the collection.

Prototypes

```
' Visual Basic

Overloads 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 class.
- **ulDbType** One of the ULDbType enumeration values.
- **size** The length of the parameter.
- **sourceColumn** The name of the source column to map.

Return value

The new ULParameter class 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 CommandText property. 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 CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "ULParameterCollection class" on page 270

- "ULParameterCollection members" on page 270
- "Add method" on page 274
- "Add method" on page 275

Clear method

Removes all the parameters from the collection.

Prototypes

' Visual Basic

```
NotOverridable Public Sub Clear() _
  Implements IList.Clear
```

// C#

```
public void Clear();
```

Implements

IList.Clear

Contains method

Checks whether a ULParameter class exists in the collection.

Prototypes

' Visual Basic

```
Overloads NotOverridable Public Function Contains( _
  ByVal value As Object _
) As Boolean _
  Implements IList.Contains
```

// C#

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

Parameters

- **value** The ULParameter class object to check for.

Return value

True if the collection contains the ULParameter class, false otherwise.

Implements

IList.Contains

See also

- "ULParameterCollection class" on page 270
- "ULParameterCollection members" on page 270
- "Contains method" on page 280

Contains method

Checks whether a ULParameter class with the specified name exists in the collection.

Prototypes

' Visual Basic

```
Overloads NotOverridable Public Function Contains( _
   ByVal value As String _
) As Boolean _
   Implements IDataParameterCollection.Contains
```

// C#

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

Parameters

- **value** The name of the parameter to search for.

Return value

True if the collection contains the ULParameter class, false otherwise.

Implements

IDataParameterCollection.Contains

See also

- "ULParameterCollection class" on page 270
- "ULParameterCollection members" on page 270
- "Contains method" on page 279

CopyTo method

Copies ULParameter class objects from the ULParameterCollection to the specified array.

Prototypes

' Visual Basic

```
NotOverridable Public Sub CopyTo( _
   ByVal array As System.Array, _
   ByVal index As Integer _
```



```
) _
  Implements ICollection.CopyTo

// C#

public void CopyTo(
  System.Array array,
  int index
);
```

Parameters

- **array** The array into which to copy the ULParameter class objects.
- **index** The starting index of the array.

Implements

ICollection.CopyTo

GetEnumerator method

Returns an enumerator for the collection.

Prototypes

```
' Visual Basic

NotOverridable Public Function GetEnumerator() As System.Collections.IEnumerator _
  Implements IEnumerable.GetEnumerator

// C#

public System.Collections.IEnumerator GetEnumerator();
```

Return value

An ArrayList enumerator enumerating the parameters in the collection.

Implements

IEnumerable.GetEnumerator

IndexOf method

Returns the location of the ULParameter class in the collection.

Prototypes

```
' Visual Basic

Overloads NotOverridable Public Function IndexOf( _
  ByVal value As Object _
) As Integer _
  Implements IList.IndexOf
```



```
// C#

public int IndexOf(
   object value
);
```

Parameters

- **value** The ULParameter class object to locate.

Return value

The zero-based index of the ULParameter class in the collection or -1 if the parameter is not found.

Exceptions

- System.InvalidCastException - The value specified must be a ULParameter class.

Implements

IList.IndexOf

See also

- "ULParameterCollection class" on page 270
- "ULParameterCollection members" on page 270
- "IndexOf method" on page 282

IndexOf method

Returns the location of the ULParameter class with the specified name in the collection.

Prototypes

```
' Visual Basic

Overloads NotOverridable Public Function IndexOf( _
   ByVal parameterName As String _
) As Integer _
   Implements IDataParameterCollection.IndexOf

// C#

public int IndexOf(
   string  parameterName
);
```

Parameters

- **parameterName** The name of the parameter to locate.

Return value

The zero-based index of the ULParameter class in the collection or -1 if the parameter is not found.

Implements

IDataParameterCollection.IndexOf

See also

- "ULParameterCollection class" on page 270
- "ULParameterCollection members" on page 270
- "IndexOf method" on page 281

Insert method

Inserts an ULParameter class in the collection at the specified index.

Prototypes

```
' Visual Basic

NotOverridable Public Sub Insert( _
   ByVal index As Integer, _
   ByVal value As Object _
) _
   Implements IList.Insert

// C#

public 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 class object to insert.

Exceptions

- System.ArgumentNullException - You cannot set a parameter using a null reference (Nothing in Visual Basic).
- System.IndexOutOfRangeException - The index is invalid.
- System.InvalidCastException - The value specified must be a ULParameter class.

Implements

IList.Insert

Remove method

Removes an ULParameter class from the collection.

Prototypes

```
' Visual Basic

NotOverridable Public Sub Remove( _
  ByVal value As Object _
) _
  Implements IList.Remove

// C#

public void Remove(
  object value
);
```

Parameters

- **value** The ULParameter class object to remove.

Exceptions

- ArgumentException - The collection does not contain the specified parameter.
- System.ArgumentNullException - You cannot set a parameter using a null reference (Nothing in Visual Basic).
- System.InvalidCastException - The value specified must be a ULParameter class.

Implements

IList.Remove

RemoveAt method

Removes the parameter at the specified index in the collection.

Prototypes

```
' Visual Basic

Overloads NotOverridable Public Sub RemoveAt( _
  ByVal index As Integer _
) _
  Implements IList.RemoveAt

// C#

public void RemoveAt(
  int index
);
```

Parameters

- **index** The zero-based index of the parameter to remove. The value must be in the range [0,Count property-1]. The first parameter in the collection has an index value of zero.

Exceptions

- System.IndexOutOfRangeException - The index is invalid.

Implements

IList.RemoveAt

See also

- “ULParameterCollection class” on page 270
- “ULParameterCollection members” on page 270
- “RemoveAt method” on page 285

RemoveAt method

Removes the parameter with the specified name from the collection.

Prototypes

```
' Visual Basic

Overloads NotOverridable Public Sub RemoveAt( _
  ByVal parameterName As String _
) _
  Implements IDataParameterCollection.RemoveAt

// C#

public void RemoveAt(
  string  parameterName
);
```

Parameters

- **parameterName** The name of the parameter to retrieve.

Exceptions

- System.IndexOutOfRangeException - There is no parameter with the specified name.

Implements

IDataParameterCollection.RemoveAt

See also

- “ULParameterCollection class” on page 270
- “ULParameterCollection members” on page 270
- “RemoveAt method” on page 284

ULPublicationSchema class

UL Ext.: Represents the schema of an UltraLite publication.

Prototypes

' Visual Basic

NotInheritable Public Class **ULPublicationSchema**

// C#

public sealed class **ULPublicationSchema**

Remarks

There is no constructor for this class. Publication schemas are created using the GetPublicationSchema method of the ULDatabaseSchema class.

UltraLite methods requiring a publication mask actually require a set of publications to check. A set is formed by or'ing the publication masks of individual publications. For example:

```
' Visual Basic
Dim mask As Integer = pub1.Mask Or pub2.Mask

// C#
int mask = pub1.Mask | pub2.Mask;
```

Two special mask values are also provided by this class. SYNC_ALL_DB field corresponds to the entire database. SYNC_ALL_PUBS field corresponds to all publications.

Note: Publication masks may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached masks after a schema upgrade.

See also

- "ULPublicationSchema members" on page 286
- "Mask property" on page 288
- "GetLastDownloadTime method" on page 102
- "CountUploadRows method" on page 96
- "Schema property" on page 90

ULPublicationSchema members

Public static fields (shared)

Member name	Description
SYNC_ALL_DB field	Publication mask corresponding to the entire database.
SYNC_ALL_PUBS field	Publication mask corresponding to all publications.

Public properties

Member name	Description
IsOpen property	Determines whether the publication schema is open or closed.
Mask property	Returns the publication mask of the publication.
Name property	Returns the name of this publication.

SYNC_ALL_DB field

Publication mask corresponding to the entire database.

Prototypes

' Visual Basic

Public Shared **SYNC_ALL_DB** As Integer

// C#

public const int **SYNC_ALL_DB**;

SYNC_ALL_PUBS field

Publication mask corresponding to all publications.

Prototypes

' Visual Basic

Public Shared **SYNC_ALL_PUBS** As Integer

// C#

public const int **SYNC_ALL_PUBS**;

IsOpen property

Determines whether the publication schema is open or closed.

Prototypes

' 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.

Mask property

Returns the publication mask of the publication.

Prototypes

' Visual Basic

Public Readonly Property **Mask** As Integer

// C#

public int **Mask** {get;}

Property value

The publication mask of the publication.

Remarks

Publication IDs, masks, and counts may change during a schema upgrade. To correctly identify a publication, access it by name, or refresh the cached masks and counts after a schema upgrade.

Exceptions

- ULException class - If a SQL error exception occurs. PUBLICATION_NOT_FOUND is issued if a schema upgrade has deleted or renamed this publication.

Name property

Returns the name of this publication.

Prototypes

' Visual Basic

Public Readonly Property **Name** As String

// C#

public string **Name** {get;}

Property value

A string specifying the name of the publication.

Exceptions

- ULException class - A SQL error occurred. PUBLICATION_NOT_FOUND is issued if a schema upgrade has deleted or renamed the publication.

ULResultSet class

UL Ext.: Represents an editable result set in an UltraLite database.

Prototypes

' 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 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 class object represents an editable result set on which you can perform positioned updates and deletes. For fully editable result sets, use ExecuteTable method or a ULDataAdapter class.

Inherits: ULDataReader class

Implements: IDataReader, IDataRecord, IDisposable

ULResultSet members

Public properties

Member name	Description
Depth property (inherited from ULDataReader)	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
FieldCount property (inherited from ULDataReader)	Returns the number of columns in the cursor.
IsBOF property (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is before the first row.

Member name	Description
IsClosed property (inherited from ULDataReader)	Checks whether the cursor is currently open.
IsEOF property (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is after the last row.
Item property (inherited from ULDataReader)	Returns the value of the specified column in its native format. In C#, this property is the indexer for the ULDataReader class.
Item property (inherited from ULDataReader)	Returns the value of the specified named column in its native format. In C#, this property is the indexer for the ULDataReader class.
RecordsAffected property (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 (inherited from ULDataReader)	**UL Ext.:** Returns the number of rows in the cursor.
Schema property (inherited from ULDataReader)	**UL Ext.:** Holds the schema of this cursor.

Public methods

Member name	Description
AppendBytes method	Appends the specified subset of the specified array of Bytes to the new value for the specified ULDbType enumeration column.
AppendChars method	Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType enumeration column.
Close method (inherited from ULDataReader)	Closes the cursor.
Delete method	Deletes the current row.
Dispose method (inherited from ULDataReader)	Releases the unmanaged resources used by the ULDataReader and optionally releases the managed resources.
GetBoolean method (inherited from ULDataReader)	Returns the value for the specified column as a Boolean.
GetByte method (inherited from ULDataReader)	Returns the value for the specified column as an unsigned 8-bit value (Byte).
GetBytes method (inherited from ULDataReader)	Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination Byte array.
GetBytes method (inherited from ULDataReader)	**UL Ext.:** Returns the value for the specified column as an array of Bytes. Only valid for columns of type ULDbType enumeration, ULDbType enumeration, or ULDbType enumeration.

Member name	Description
GetChar method (inherited from ULDataReader)	This method is not supported in UltraLite.NET.
GetChars method (inherited from ULDataReader)	Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination System.Char array.
GetData method (inherited from ULDataReader)	This method is not supported in UltraLite.NET.
GetDataTypeName method (inherited from ULDataReader)	Returns the name of the specified column's provider data type.
GetDateTime method (inherited from ULDataReader)	Returns the value for the specified column as a DateTime with millisecond accuracy.
GetDecimal method (inherited from ULDataReader)	Returns the value for the specified column as a Decimal.
GetDouble method (inherited from ULDataReader)	Returns the value for the specified column as a Double.
GetFieldType method (inherited from ULDataReader)	Returns the Type most appropriate for the specified column.
GetFloat method (inherited from ULDataReader)	Returns the value for the specified column as a Single.
GetGuid method (inherited from ULDataReader)	Returns the value for the specified column as a UUID (Guid).
GetInt16 method (inherited from ULDataReader)	Returns the value for the specified column as an Int16.
GetInt32 method (inherited from ULDataReader)	Returns the value for the specified column as an Int32.
GetInt64 method (inherited from ULDataReader)	Returns the value for the specified column as an Int64.
GetName method (inherited from ULDataReader)	Returns the name of the specified column.
GetOrdinal method (inherited from ULDataReader)	Returns the column ID of the named column.
GetSchemaTable method (inherited from ULDataReader)	Returns a DataTable that describes the column metadata of the ULDataReader.
GetString method (inherited from ULDataReader)	Returns the value for the specified column as a String.

Member name	Description
GetTimeSpan method (inherited from ULDataReader)	Returns the value for the specified column as a TimeSpan with millisecond accuracy.
GetUInt16 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt16.
GetUInt32 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt32.
GetUInt64 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt64.
GetValue method (inherited from ULDataReader)	Returns the value of the specified column in its native format.
GetValues method (inherited from ULDataReader)	Returns all the column values for the current row.
IsDBNull method (inherited from ULDataReader)	Checks whether the value from the specified column is NULL.
MoveAfterLast method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to after the last row of the cursor.
MoveBeforeFirst method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to before the first row of the cursor.
MoveFirst method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the first row of the cursor.
MoveLast method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the last row of the cursor.
MoveNext method (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 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the previous row or before the first row.
MoveRelative method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor relative to the current row.
NextResult method (inherited from ULDataReader)	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
Read method (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	Sets the value for the specified column using a Boolean.
SetByte method	Sets the value for the specified column using a Byte (unsigned 8-bit integer).

Member name	Description
SetBytes method	Sets the value for the specified column using an array of Bytes.
SetDateTime method	Sets the value for the specified column using a DateTime.
SetDBNull method	Sets a column to NULL.
SetDecimal method	Sets the value for the specified column using a Decimal.
SetDouble method	Sets the value for the specified column using a Double.
SetFloat method	Sets the value for the specified column using a Single.
SetGuid method	Sets the value for the specified column using a Guid.
SetInt16 method	Sets the value for the specified column using an Int16.
SetInt32 method	Sets the value for the specified column using an Int32.
SetInt64 method	Sets the value for the specified column using an Int64.
SetString method	Sets the value for the specified column using a String.
SetTimeSpan method	Sets the value for the specified column using a TimeSpan.
SetToDefault method	Sets the value for the specified column to its default value.
SetUInt16 method	Sets the value for the specified column using a UInt16.
SetUInt32 method	Sets the value for the specified column using an UInt32.
SetUInt64 method	Sets the value for the specified column using a UInt64.
Update method	Updates the current row with the current column values (specified using the set methods).

Protected methods

Member name	Description
Finalize method	Releases unmanaged resources and performs other cleanup operations before the ULResultSet is reclaimed by garbage collection.

AppendBytes method

Appends the specified subset of the specified array of Bytes to the new value for the specified ULDbType enumeration column.

Prototypes

' 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,FieldCount property-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, InsertBegin method initializes the new value to the column's default value. The data in the row is not actually changed until you execute an Insert method, 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 class with code ULSQLCode enumeration 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 class with the appropriate error code is thrown.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

AppendChars method

Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType enumeration column.

Prototypes

' 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,FieldCount property-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, InsertBegin method initializes the new value to the column's default value. The data in the row is not actually changed until you execute an Insert method, 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 class with code ULSQLCode enumeration 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 class with the appropriate error code is thrown.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

Delete method

Deletes the current row.

Prototypes

' Visual Basic

Public Sub **Delete()**

// C#

public void **Delete();**

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "StartSynchronizationDelete method" on page 106
- "StopSynchronizationDelete method" on page 107

Finalize method

Releases unmanaged resources and performs other cleanup operations before the ULResultSet is reclaimed by garbage collection.

Prototypes

' Visual Basic

Overrides Protected Sub **Finalize()**

// C#

protected override void **Finalize();**

SetBoolean method

Sets the value for the specified column using a Boolean.

Prototypes

' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULResultSet class” on page 289
- “ULResultSet members” on page 289
- “GetOrdinal method” on page 203
- “Schema property” on page 186
- “GetFieldType method” on page 198

SetByte method

Sets the value for the specified column using a Byte (unsigned 8-bit integer).

Prototypes

' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetBytes method

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

Prototypes

' 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,FieldCount property-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 enumeration or ULDbType enumeration, or for columns of type ULDbType enumeration when val is of length 16. The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetDateTime method

Sets the value for the specified column using a DateTime.

Prototypes

```
' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetDBNull method

Sets a column to NULL.

Prototypes

```
' 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,FieldCount property-1]. The first column in the cursor has an ID value of zero.

Remarks

The data is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "IsColumnNullable method" on page 421

SetDecimal method

Sets the value for the specified column using a Decimal.

Prototypes

```
' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetDouble method

Sets the value for the specified column using a Double.

Prototypes

' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetFloat method

Sets the value for the specified column using a Single.

Prototypes

' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetGuid method

Sets the value for the specified column using a Guid.

Prototypes

```
' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed. Only valid for columns of type ULDbType enumeration or for columns of type ULDbType enumeration with length 16.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetNewUUID method" on page 103
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198
- "GetColumnSize method" on page 137

SetInt16 method

Sets the value for the specified column using an Int16.

Prototypes

```
' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetInt32 method

Sets the value for the specified column using an Int32.

Prototypes

```
' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289

- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetInt64 method

Sets the value for the specified column using an Int64.

Prototypes

```
' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetString method

Sets the value for the specified column using a String.

Prototypes

' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULResultSet class” on page 289
- “ULResultSet members” on page 289
- “GetOrdinal method” on page 203
- “Schema property” on page 186
- “GetFieldType method” on page 198

SetTimeSpan method

Sets the value for the specified column using a TimeSpan.

Prototypes

' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetToDefault method

Sets the value for the specified column to its default value.

Prototypes

```
' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198
- "GetColumnDefaultValue method" on page 414

SetUInt16 method

Sets the value for the specified column using a UInt16.

Prototypes

' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "GetFieldType method" on page 198

SetUInt32 method

Sets the value for the specified column using an UInt32.

Prototypes

```
' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

SetUInt64 method

Sets the value for the specified column using a UInt64.

Prototypes

' 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,FieldCount property-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 Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULResultSet class" on page 289
- "ULResultSet members" on page 289
- "GetOrdinal method" on page 203
- "Schema property" on page 186
- "GetFieldType method" on page 198

Update method

Updates the current row with the current column values (specified using the set methods).

Prototypes

' Visual Basic

```
Public Sub Update()
```



```
// C#

public void Update();
```

Exceptions

- ULException class - A SQL error occurred.

ULResultSetSchema class

UL Ext.: Represents the schema of an UltraLite result set.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULResultSetSchema
  Inherits ULCursorSchema
```

// C#

```
public sealed class ULResultSetSchema :
  ULCursorSchema
```

Remarks

There is no constructor for this class. A ULResultSetSchema class object is attached to a result set as its Schema property.

A result set schema is only valid while the data reader is open.

Inherits: ULCursorSchema class

See also

- "ULResultSetSchema members" on page 313
- "ULCommand class" on page 54
- "ULDataReader class" on page 178

ULResultSetSchema members

Public properties

Member name	Description
ColumnCount property (inherited from ULCursorSchema)	Returns the number of columns in the cursor.
IsOpen property (inherited from ULCursorSchema)	Checks whether the cursor schema is currently open.
Name property	Returns the name of the cursor.

Public methods

Member name	Description
GetColumnID method (inherited from ULCursorSchema)	Returns the column ID of the named column.

Member name	Description
GetColumnName method (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
GetColumnPrecision method (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).
GetColumnScale method (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 (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).
GetColumnSQLName method (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
GetColumnULDbType method (inherited from ULCursorSchema)	Returns the UltraLite.NET data type of the column identified by the specified column ID.
GetSchemaTable method (inherited from ULCursorSchema)	Returns a DataTable that describes the column schema of the ULDataReader class.

Name property

Returns the name of the cursor.

Prototypes

' Visual Basic

Public Readonly Property **Name** As String

// C#

public string **Name** {get;}

Property value

The SQL statement that generated the ULResultSetSchema.

ULRowUpdatedEventArgs class

Provides data for the RowUpdated event event.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULRowUpdatedEventArgs
  Inherits RowUpdatedEventArgs
```

// C#

```
public sealed class ULRowUpdatedEventArgs :
  RowUpdatedEventArgs
```

Remarks

Inherits: RowUpdatedEventArgs

ULRowUpdatedEventArgs members

Public constructors

Member name	Description
ULRowUpdatedEventArgs constructor	Initializes a new instance of the ULRowUpdatedEventArgs class.

Public properties

Member name	Description
Command property	Returns the ULCommand class executed when DbDataAdapter.Update is called.
Errors (inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the RowUpdatedEventArgs.Command was executed.
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.
Row (inherited from RowUpdatedEventArgs)	Gets the DataRow sent through an DbDataAdapter.Update .
StatementType (inherited from RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status (inherited from RowUpdatedEventArgs)	Gets the UpdateStatus of the RowUpdatedEventArgs.Command property.

Member name	Description
TableMapping (inherited from RowUpdatedEventArgs)	Gets the DataTableMapping sent through an DbDataAdapter.Update.

ULRowUpdatedEventArgs constructor

Initializes a new instance of the ULRowUpdatedEventArgs class.

Prototypes

' 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 DataRow sent through an DbDataAdapter.Update.
- **command** The IDbCommand executed when DbDataAdapter.Update is called.
- **statementType** One of the StatementType values that specifies the type of query executed.
- **tableMapping** The DataTableMapping sent through an DbDataAdapter.Update.

Command property

Returns the ULCommand class executed when DbDataAdapter.Update is called.

Prototypes

' Visual Basic

```
Public Readonly Property Command As ULCommand
```

// C#

```
public ULCommand Command {get;}
```


Property value

The ULCommand class object executed by the update.

Remarks

This is the strongly-typed version of 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.

Prototypes

' 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.

ULRowUpdatedEventHandler delegate

Represents the method that will handle the RowUpdated event event.

Prototypes

```
' Visual Basic

Public Delegate Sub ULRowUpdatedEventHandler( _
   ByVal sender As Object, _
   ByVal e As ULRowUpdatedEventArgs _
)

// C#

public delegate void ULRowUpdatedEventHandler(
   object sender,
   ULRowUpdatedEventArgs e
);
```

Parameters

- **sender** The connection sending the event.
- **e** The ULRowUpdatedEventArgs class object that contains the event data.

ULRowUpdatingEventArgs class

Provides data for the RowUpdating event event.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULRowUpdatingEventArgs
  Inherits RowUpdatingEventArgs
```

// C#

```
public sealed class ULRowUpdatingEventArgs :
  RowUpdatingEventArgs
```

Remarks

Inherits: RowUpdatingEventArgs

ULRowUpdatingEventArgs members

Public constructors

Member name	Description
ULRowUpdatingEventArgs constructor	Initializes a new instance of the ULRowUpdatingEventArgs class.

Public properties

Member name	Description
Command property	Specifies the ULCommand class to execute when performing the DbDataAdapter.Update.
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 to send through an DbDataAdapter.Update .
StatementType (inherited from RowUpdatingEventArgs)	Gets the type of SQL statement to execute.
Status (inherited from RowUpdatingEventArgs)	Gets the UpdateStatus of the RowUpdatedEventArgs.Command property.
TableMapping (inherited from RowUpdatingEventArgs)	Gets the DataTableMapping to send through the DbDataAdapter.Update.

ULRowUpdatingEventArgs constructor

Initializes a new instance of the ULRowUpdatingEventArgs class.

Prototypes

' 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 DataRow to update.
- **command** The IDbCommand to execute during the update.
- **statementType** One of the StatementType values that specifies the type of query executed.
- **tableMapping** The DataTableMapping sent through an DbDataAdapter.Update.

Command property

Specifies the ULCommand class to execute when performing the DbDataAdapter.Update.

Prototypes

' Visual Basic

```
Public Property Command As ULCommand
```

// C#

```
public ULCommand Command {get;set;}
```

Property value

The ULCommand class object to execute when updating.

Remarks

This is the strongly-typed version of RowUpdatingEventArgs.Command.

ULRowUpdatingEventHandler delegate

Represents the method that will handle the RowUpdating event event.

Prototypes

```
' Visual Basic

Public Delegate Sub ULRowUpdatingEventHandler( _
   ByVal sender As Object, _
   ByVal e As ULRowUpdatingEventArgs _
)

// C#

public delegate void ULRowUpdatingEventHandler(
   object sender,
   ULRowUpdatingEventArgs e
);
```

Parameters

- **sender** The connection sending the event.
- **e** The ULRowUpdatingEventArgs class object that contains the event data.

ULRuntimeType enumeration

UL Ext.: Enumerates the types of UltraLite.NET runtimes.

Prototypes

' Visual Basic

```
Public Enum ULRuntimeType
  Inherits Short
```

// C#

```
public enum ULRuntimeType :
  short
```

Member name

Member name	Description
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.
UL_ENGINE_CLIENT	Selects the UltraLite engine runtime. The UltraLite.NET engine client communicates with the UltraLite engine to access databases. This means that databases can be shared by different applications.

See also

- "RuntimeType property" on page 162

ULServerSyncListener interface

UL Ext.: The listener interface for receiving server synchronization messages.

Prototypes

' Visual Basic

Public Interface **ULServerSyncListener**

// C#

public interface **ULServerSyncListener**

ULServerSyncListener members

Public methods

Member name	Description
ServerSyncInvoked method	Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

ServerSyncInvoked method

Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

Prototypes

' 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();
    }
  }
```

ULSQLCode enumeration

UL Ext.: Enumerates the SQL codes that may be reported by UltraLite.NET.

Prototypes

' Visual Basic

Public Enum **ULSQLCode**

// C#

public enum **ULSQLCode**

Member name

Member name	Description
SQLE_AGGREGATES_NOT_ALLOWED	See "Invalid use of an aggregate function" [*SQL Anywhere 10 - Error Messages*].
SQLE_ALIAS_NOT_UNIQUE	See "Alias '%1' is not unique" [*SQL Anywhere 10 - Error Messages*].
SQLE_ALIAS_NOT_YET_DEFINED	See "Definition for alias '%1' must appear before its first reference" [*SQL Anywhere 10 - Error Messages*].
SQLE_AMBIGUOUS_INDEX_NAME	See "Index name '%1' is ambiguous" [*SQL Anywhere 10 - Error Messages*].
SQLE_ARGUMENT_CANNOT_BE_NULL	See "Argument %1 of procedure '%2' cannot be NULL" [*SQL Anywhere 10 - Error Messages*].
SQLE_BAD_ENCRYPTION_KEY	See "Incorrect or missing encryption key" [*SQL Anywhere 10 - Error Messages*].
SQLE_BAD_PARAM_INDEX	See "Input parameter index out of range" [*SQL Anywhere 10 - Error Messages*].
SQLE_CANNOT_ACCESS_FILESYSTEM	See "Unable to access the filesystem on the device" [*SQL Anywhere 10 - Error Messages*].
SQLE_CANNOT_CHANGE_ML_REMOTE_ID	See "Cannot change the MobiLink remote id when the status of the last upload is unknown" [*SQL Anywhere 10 - Error Messages*].
SQLE_CANNOT_CONVERT	See "Invalid data conversion" [*SQL Anywhere 10 - Error Messages*].
SQLE_CANNOT_EXECUTE_STMT	See "Statement cannot be executed" [*SQL Anywhere 10 - Error Messages*].
SQLE_CANNOT_MODIFY	See "Cannot modify column '%1' in table '%2'" [*SQL Anywhere 10 - Error Messages*].

Member name	Description
SQLE_CANNOT_REGISTER_LISTENER	See “The specified listener could not be registered” [*SQL Anywhere 10 - Error Messages*].
SQLE_CLIENT_OUT_OF_MEMORY	See “Client out of memory” [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_AMBIGUOUS	See “Column '%1' found in more than one table -- need a correlation name” [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_CANNOT_BE_NULL	See “Column '%1' in table '%2' cannot be NULL” [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_IN_INDEX	See “Cannot alter a column in an index” [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_NOT_FOUND	See “Column '%1' not found” [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_NOT_INDEXED	See “Column '%1' not part of any indexes in its containing table” [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_NOT_STREAMABLE	See “The operation failed because column '%1"s type does not support streaming” [*SQL Anywhere 10 - Error Messages*].
SQLE_COMMUNICATIONS_ERROR	See “Communication error” [*SQL Anywhere 10 - Error Messages*].
SQLE_CONNECTION_ALREADY_EXISTS	See “This connection already exists” [*SQL Anywhere 10 - Error Messages*].
SQLE_CONNECTION_NOT_FOUND	See “Connection not found” [*SQL Anywhere 10 - Error Messages*].
SQLE_CONNECTION_RESTORED	See “UltraLite connection was restored” [*SQL Anywhere 10 - Error Messages*].
SQLE_CONSTRAINT_NOT_FOUND	See “Constraint '%1' not found” [*SQL Anywhere 10 - Error Messages*].
SQLE_CONVERSION_ERROR	See “Cannot convert %1 to a %2” [*SQL Anywhere 10 - Error Messages*].
SQLE_COULD_NOT_FIND_FUNCTION	See “Could not find '%1' in dynamic library '%2'” [*SQL Anywhere 10 - Error Messages*].
SQLE_COULD_NOT_LOAD_LIBRARY	See “Could not load dynamic library '%1'” [*SQL Anywhere 10 - Error Messages*].
SQLE_CURSOR_ALREADY_OPEN	See “Cursor already open” [*SQL Anywhere 10 - Error Messages*].
SQLE_CURSOR_NOT_DECLARED	See “Cursor has not been declared” [*SQL Anywhere 10 - Error Messages*].

Member name	Description
SQLE_CURSOR_NOT_OPEN	See “Cursor not open” [*SQL Anywhere 10 - Error Messages*].
SQLE_CURSOR_RESTORED	See “UltraLite cursor (or result set or table) was restored” [*SQL Anywhere 10 - Error Messages*].
SQLE_CURSOROP_NOT_ALLOWED	See “Illegal cursor operation attempt” [*SQL Anywhere 10 - Error Messages*].
SQLE_DATABASE_ERROR	See “Internal database error %1 -- transaction rolled back” [*SQL Anywhere 10 - Error Messages*].
SQLE_DATABASE_NAME_REQUIRED	See “Database name required to start server” [*SQL Anywhere 10 - Error Messages*].
SQLE_DATABASE_NOT_CREATED	See “Database creation failed: %1” [*SQL Anywhere 10 - Error Messages*].
SQLE_DATATYPE_NOT_ALLOWED	See “Expression has unsupported data type” [*SQL Anywhere 10 - Error Messages*].
SQLE_DBSPACE_FULL	See “A dbspace has reached its maximum file size” [*SQL Anywhere 10 - Error Messages*].
SQLE_DESCRIBE_NONSELECT	See “Can only describe a SELECT statement” [*SQL Anywhere 10 - Error Messages*].
SQLE_DEVICE_IO_FAILED	See “File I/O failed for '%1'” [*SQL Anywhere 10 - Error Messages*].
SQLE_DIV_ZERO_ERROR	See “Division by zero” [*SQL Anywhere 10 - Error Messages*].
SQLE_DOWNLOAD_CONFLICT	See “Download failed because of conflicts with existing rows” [*SQL Anywhere 10 - Error Messages*].
SQLE_DOWNLOAD_RESTART_FAILED	See “Unable to retry download because upload is not finished” [*SQL Anywhere 10 - Error Messages*].
SQLE_DROP_DATABASE_FAILED	See “An attempt to delete database '%1' failed” [*SQL Anywhere 10 - Error Messages*].
SQLE_DUPLICATE_CURSOR_NAME	See “The cursor name '%1' already exists” [*SQL Anywhere 10 - Error Messages*].
SQLE_DUPLICATE_FOREIGN_KEY	See “Foreign key '%1' for table '%2' duplicates an existing foreign key” [*SQL Anywhere 10 - Error Messages*].
SQLE_DUPLICATE_OPTION	See “Option '%1' specified more than once” [*SQL Anywhere 10 - Error Messages*].
SQLE_DYNAMIC_MEMORY_EXHAUSTED	See “Dynamic memory exhausted” [*SQL Anywhere 10 - Error Messages*].
SQLE_ENCRYPTION_INITIALIZATION_FAILED	See “Could not initialize the encryption DLL: '%1'” [*SQL Anywhere 10 - Error Messages*].

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

Member name	Description
SQLE_INVALID_CONSTRAINT_REF	See “Invalid reference to or operation on constraint '%1'” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_DESCRIPTOR_INDEX	See “Invalid descriptor index” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_DESCRIPTOR_NAME	See “Invalid SQL descriptor name” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_DISTINCT_AGGREGATE	See “Grouped query contains more than one distinct aggregate function” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_FOREIGN_KEY	See “No primary key value for foreign key '%1' in table '%2'” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_FOREIGN_KEY_DEF	See “Column '%1' in foreign key has a different definition than primary key” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_GROUP_SELECT	See “Function or column reference to '%1' must also appear in a GROUP BY” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_INDEX_TYPE	See “Index type specification of '%1' is invalid” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_LOGON	See “Invalid user ID or password” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_OPTION_SETTING	See “Invalid setting for option '%1'” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_OPTION_VALUE	See “'%1' is an invalid value for '%2'” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_ORDER	See “Invalid ORDER BY specification” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_PARAMETER	See “Invalid parameter” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_PARSE_PARAMETER	See “Parse error: %1” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_PUBLICATION_MASK	See “The specified publication mask is invalid” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_SQL_IDENTIFIER	See “Invalid SQL identifier” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_STATEMENT	See “Invalid statement” [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_UNION	See “Select lists in UNION, INTERSECT, or EXCEPT do not match in length” [*SQL Anywhere 10 - Error Messages*].

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

Member name	Description
SQLE_PAGE_SIZE_INVALID	See “Invalid database page size” [*SQL Anywhere 10 - Error Messages*].
SQLE_PARTIAL_DOWNLOAD_NOT_FOUND	See “No partial download was found” [*SQL Anywhere 10 - Error Messages*].
SQLE_PERMISSION_DENIED	See “Permission denied: %1” [*SQL Anywhere 10 - Error Messages*].
SQLE_PRIMARY_KEY_NOT_UNIQUE	See “Primary key for table '%1' is not unique” [*SQL Anywhere 10 - Error Messages*].
SQLE_PRIMARY_KEY_TWICE	See “Table cannot have two primary keys” [*SQL Anywhere 10 - Error Messages*].
SQLE_PRIMARY_KEY_VALUE_REF	See “Primary key for row in table '%1' is referenced by foreign key '%2' in table '%3'” [*SQL Anywhere 10 - Error Messages*].
SQLE_PUBLICATION_NOT_FOUND	See “Publication '%1' not found” [*SQL Anywhere 10 - Error Messages*].
SQLE_PUBLICATION_PREDICATE_IGNORED	See “Publication predicates were not evaluated” [*SQL Anywhere 10 - Error Messages*].
SQLE_RESOURCE_GOVERNOR_EXCEEDED	See “Resource governor for '%1' exceeded” [*SQL Anywhere 10 - Error Messages*].
SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY	See “Row was dropped from table %1 to maintain referential integrity” [*SQL Anywhere 10 - Error Messages*].
SQLE_ROW_EXCEEDS_PAGE_SIZE	See “A row cannot be stored because it exceeds the database page size” [*SQL Anywhere 10 - Error Messages*].
SQLE_SCHEMA_UPGRADE_NOT_ALLOWED	See “A schema upgrade is not currently allowed” [*SQL Anywhere 10 - Error Messages*].
SQLE_SERVER_SYNCHRONIZATION_ERROR	See “Synchronization failed due to an error on the server: %1” [*SQL Anywhere 10 - Error Messages*].
SQLE_START_STOP_DATABASE_DENIED	See “Request to start/stop database denied” [*SQL Anywhere 10 - Error Messages*].
SQLE_STATEMENT_ERROR	See “SQL statement error” [*SQL Anywhere 10 - Error Messages*].
SQLE_STRING_RIGHT_TRUNCATION	See “Right truncation of string data” [*SQL Anywhere 10 - Error Messages*].
SQLE_SUBQUERY_SELECT_LIST	See “Subquery allowed only one select list item” [*SQL Anywhere 10 - Error Messages*].
SQLE_SYNC_INFO_INVALID	See “Information for synchronization is incomplete or invalid, check '%1'” [*SQL Anywhere 10 - Error Messages*].

Member name	Description
SQLE_SYNC_INFO_REQUIRED	See “Information for synchronization was not provided” [*SQL Anywhere 10 - Error Messages*].
SQLE_SYNC_NOT_REENTRANT	See “Synchronization process was unable to re-enter synchronization” [*SQL Anywhere 10 - Error Messages*].
SQLE_SYNC_STATUS_UNKNOWN	See “The status of the last synchronization upload is unknown” [*SQL Anywhere 10 - Error Messages*].
SQLE_SYNTAX_ERROR	See “Syntax error near '%1' %2” [*SQL Anywhere 10 - Error Messages*].
SQLE_TABLE_ALREADY_INCLUDED	See “Table '%1' is already included” [*SQL Anywhere 10 - Error Messages*].
SQLE_TABLE_IN_USE	See “Table in use” [*SQL Anywhere 10 - Error Messages*].
SQLE_TABLE_NOT_FOUND	See “Table '%1' not found” [*SQL Anywhere 10 - Error Messages*].
SQLE_TOO_MANY_BLOB_REFS	See “Too many references to a BLOB” [*SQL Anywhere 10 - Error Messages*].
SQLE_TOO_MANY_CONNECTIONS	See “Database server connection limit exceeded” [*SQL Anywhere 10 - Error Messages*].
SQLE_TOO_MANY_PUBLICATIONS	See “Too many publications specified in publication mask” [*SQL Anywhere 10 - Error Messages*].
SQLE_TOO_MANY_TEMP_TABLES	See “Too many temporary tables in connection” [*SQL Anywhere 10 - Error Messages*].
SQLE_TOO_MANY_USERS	See “Too many users in database” [*SQL Anywhere 10 - Error Messages*].
SQLE_ULTRALITE_DATABASE_NOT_FOUND	See “The database '%1' was not found” [*SQL Anywhere 10 - Error Messages*].
SQLE_ULTRALITE_OBJ_CLOSED	See “Invalid operation on a closed object” [*SQL Anywhere 10 - Error Messages*].
SQLE_ULTRALITE_WRITE_ACCESS_DENIED	See “Write access was denied” [*SQL Anywhere 10 - Error Messages*].
SQLE_UNABLE_TO_CONNECT	See “Database cannot be started -- %1” [*SQL Anywhere 10 - Error Messages*].
SQLE_UNABLE_TO_START_DATABASE	See “Unable to start specified database: %1” [*SQL Anywhere 10 - Error Messages*].
SQLE_UNABLE_TO_START_DATABASE_VER_NEWER	See “Unable to start specified database: Server must be upgraded to start database %1” [*SQL Anywhere 10 - Error Messages*].

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

ULStreamErrorCode enumeration

UL Ext.: Enumerates the error codes that may be reported by streams during synchronization.

Prototypes

' Visual Basic

```
Public Enum ULStreamErrorCode
  Inherits Short
```

// C#

```
public enum ULStreamErrorCode :
  short
```

Member name

Member name	Description
ACTSYNC_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.
ACTSYNC_NOT_INSTALLED	The ActiveSync provider has not been installed. Run dbasinst to install it (see documentation for details).
CONNECT_TIMEOUT	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.
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.
CREATE_RANDOM_OBJECT	The secure network layer could not create a random-number-generating object. Free up system resources, reconnect and retry the operation.
DEQUEUING_CONNECTION	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.
DUN_DIAL_FAILED	Automatic dialup failed to establish connection to the specified dial up network.
DUN_NOT_SUPPORTED	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 above. Dialup is not supported on other platforms.
END_READ	Unable to finish a sequence of reads from the network. See also: READ

Member name	Description
END_WRITE	Unable to finish a sequence of writes to the network. See also: WRITE
GENERATE_RANDOM	The secure network layer requires a random number but was unable to generate one. Free up system resources, reconnect and retry the operation.
HTTP_AUTHENTICATION_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.
HTTP_AUTHENTICATION_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.
HTTP_BAD_STATUS_CODE	Examine the status line to determine the cause of the failure.
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.
HTTP_CHUNK_LEN_BAD_CHARACTER	Try using a fixed length HTTP body.
HTTP_CHUNK_LEN_ENCODED_MISSING	Try using a fixed length HTTP body.
HTTP_CLIENT_ID_NOT_SET	The client id was not passed into the HTTP client code. Contact technical support for a fix.
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.
HTTP_CRLF_ENCODED_MISSING	The proxy you are using may not be compatible with MobiLink. Please check your configuration.
HTTP_CRLF_MISSING	The proxy you are using may not be compatible with MobiLink. Please check your configuration.
HTTP_EXPECTED_POST	The proxy you are using may not be compatible with MobiLink. Please check your configuration.
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.
HTTP_HEADER_PARSE_ERROR	An error occurred while trying to parse an HTTP header. The header may be malformed.
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.
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.

Member name	Description
HTTP_INVALID_CHARAC-TER	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.
HTTP_INVALID_SESSION_K EY	An unknown session key type was specified. Refer to the documentation and change the session key type to one of the supported types.
HTTP_MALFORMED_SESSI ON_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,
HTTP_NO_CONTD_CON-NECTION	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.
HTTP_NO_PASSWORD	A userid was supplied for HTTP authentication but no password. Both are required for authentication.
HTTP_NO_USERID	A password was supplied for HTTP authentication but no userid. Both are required for authentication.
HTTP_PROXY_AUTHENTIC ATION_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.
HTTP_PROXY_AUTHENTIC ATION_REQUIRED	An HTTP proxy requested HTTP authentication. Please supply a userid and password using the HTTP synchronization parameters http_proxy_userid and http_proxy_password.
HTTP_SERVER_AUTH_FAIL ED	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.
HTTP_UNABLE_TO_PARSE_ COOKIE	Determine where the set cookie header is being corrupted.
HTTP_UNKNOWN_TRANSF ER_ENCODING	Determine how the unknown transfer encoding is getting generated.
HTTP_UNSUPPORTED_AUT H_ALGORITHM	The HTTP Digest authentication algorithm requested by the server is unsupported. Only "MD5" and "MD5-sess" are supported.
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.
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.
INIT_RANDOM	The secure network layer could not initialize its random number generator. Free up system resources, reconnect and retry the operation.

Member name	Description
INTERNAL	An internal error has occurred in the network layer. Please contact technical support.
INTERNAL_API	An internal error has occurred in the network layer. Please contact technical support.
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.
INTERRUPTED	The current operation was interrupted by the caller.
INVALID_COMPRESSION_TYPE	The specified compression type was not recognized.
INVALID_LOCAL_PATH	The local path for the downloaded file is invalid. Consult the documentation for details.
INVALID_SYNC_PROTOCOL	The specified protocol is not a valid synchronization protocol.
LIBRARY_ENTRY_POINT_NOT_FOUND	The indicated library entry point could not be found.
LOAD_LIBRARY_FAILURE	The indicated library could not be found in the path. If you are trying to use TLS encryption for synchronization, make sure you have acquired the proper license.
LOAD_NETWORK_LIBRARY	The network interface library could not be found and/or loaded. Please check the following: 1) The sockets layer is properly installed. The correct 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.
MEMORY_ALLOCATION	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 operating system documentation for details.
MISSING_PARAMETER	The specified parameter was expected but not supplied.
NO_ECC_FIPS	There was a problem performing the given compression operation. Please contact technical support.
NONE	This code indicates there was either no network error, or an unknown network error occurred.
NOT_IMPLEMENTED	An unimplemented internal feature was requested. Please contact technical support.

Member name	Description
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.
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).
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.
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.
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.
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.
PROTOCOL_ERROR	An unexpected value or token was read.

Member name	Description
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 output 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 output 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.
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.
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.
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.
SECURE_CERTIFICATE_CHAIN_FUNC_XXX	[Warning: missing documentation]
SECURE_CERTIFICATE_CHAIN_LENGTH_XXX	[Warning: missing documentation]
SECURE_CERTIFICATE_CHAIN_REF_XXX	[Warning: missing documentation]
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 readcert utility.

Member name	Description
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 readcert utility.
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 readcert utility.
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).
SECURE_CERTIFICATE_EXPIRED	A certificate in the certificate chain has expired. Obtain a new certificate with a later expiry date and retry the operation.
SECURE_CERTIFICATE_EXPIRY_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 certificates. 3) The certificate file contains the correct certificate(s). 4) The certificate file is undamaged.
SECURE_CERTIFICATE_FILE_NOT_FOUND	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 certificates. 4) The certificate file contains the correct certificate(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.

Member name	Description
SECURE_CERTIFICATE_NOT_TRUSTED	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 certificates. 3) The certificate file contains the correct certificate(s). 4) The client's list of trusted root certificates includes the server's root certificate.
SECURE_CERTIFICATE_REF_XXX	[Warning: missing documentation]
SECURE_CERTIFICATE_ROOT	The root certificate in the chain is invalid. At the time of publication, this error was defined but not used.
SECURE_CREATE_CERTIFICATE	The secure network layer was unable to allocate storage for a certificate. Free up system resources and retry the operation.
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.
SECURE_DUPLICATE_CONTEXT	The secure network layer was unable to duplicate a security context. Free up system resources and retry the operation.
SECURE_ENABLE_NON_BLOCKING_XXX	[Warning: missing documentation]
SECURE_EXPORT_CERTIFICATE	The secure network layer was unable to copy a certificate. Free up system resources and retry the operation.
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 certificate file was specified, both on the client and on the server.
SECURE_IMPORT_CERT_FROM_SYSTEM_STORE	Failed to import a certificate from the system certificate store.
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).

Member name	Description
SECURE_NO_CERTS_IN_SYS_STORE	No certificates were found in the system's certificate store.
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 as well as its private key.
SECURE_NO_SERVER_CERTIFICATE_PASSWORD	No server certificate password was provided. This password is required to decrypt the server's encrypted private key.
SECURE_NO_TRUSTED_ROOTS	No trusted root certificates were provided. At least one trusted root certificate is required for secure communications.
SECURE_OPEN_SYSTEM_CERT_STORE	An attempt to open a system certificate store failed.
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.
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.
SECURE_REDUNDANT_SERVER_CERTIFICATE_PASSWORD	A password was specified when the server's private key wasn't encrypted by any password.
SECURE_SET_CHAIN_NUMBER_XXX	[Warning: missing documentation]
SECURE_SET_CIPHER_SUITES_XXX	[Warning: missing documentation]
SECURE_SET_IO	The secure network layer was unable to attach to the network layer. Free up system resources and retry the operation.
SECURE_SET_IO_SEMANTICS_XXX	[Warning: missing documentation]

Member name	Description
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.
SECURE_SET_PROTOCOL_SIDE_XXX	[Warning: missing documentation]
SECURE_SET_RANDOM_FUNC_XXX	[Warning: missing documentation]
SECURE_SET_RANDOM_REF_XXX	[Warning: missing documentation]
SECURE_SET_READ_FUNC	[Warning: missing documentation]
SECURE_SET_WRITE_FUNC	[Warning: missing documentation]
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.
SECURE_TRUSTED_CERTIFICATE_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 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.
SEED_RANDOM	The secure network layer could not seed its random number generator. Free up system resources, reconnect and retry the operation.
SERVER_ERROR	The server reported an error. Contact the MobiLink administrator to learn more.

Member name	Description
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.
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.
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.
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.

Member name	Description
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.
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.
SOCKET_CREATE_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.

Member name	Description
SOCKET_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.
SOCKET_GET_NAME	The network layer was unable to determine a socket'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 connection 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 normally. 3) There are enough system resources available. Free up system resources if they are running low.
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.
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.
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.

Member name	Description
SOCKET_LIVENESS_OUT_OF_RANGE	An invalid liveness timeout value was specified. The liveness timeout value must be an integer between zero and 65535.
SOCKET_LOCALHOST_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.
SOCKET_PORT_OUT_OF_RANGE	An invalid port number was specified. The port number must be an integer between zero and 65535.
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.
SOCKET_SET_OPTION	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.
SOCKET_SHUTDOWN	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.

Member name	Description
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.
UNEXPECTED_HTTP_REQUEST_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.
UNRECOGNIZED_TLS_TYPE	The TLS type is invalid. Consult the documentation for valid types.
VALUE_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.
WOULD_BLOCK	A requested operation would block where blocking is undesirable or unexpected.
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 output 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 output 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.
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.

See also

- "StreamErrorCode property" on page 385

ULStreamErrorContext enumeration

UL Ext.: Enumerates the basic network operation being performed when the stream errors occured.

Prototypes

' Visual Basic

```
Public Enum ULStreamErrorContext
  Inherits Short
```

// C#

```
public enum ULStreamErrorContext :
  short
```

Member name

Member name	Description
CLOSE	CLOSE(6)
CREATE	CREATE(3)
DESTROY	DESTROY(4)
END_READ	END_READ(11)
END_WRITE	END_WRITE(10)
GETVALUE	GETVALUE(14)
OPEN	OPEN(5)
READ	READ(7)
REGISTER	REGISTER(1)
SOFTSHUTDOWN	SOFTSHUTDOWN(13)
UNKNOWN	UNKNOWN(0)
UNREGISTER	UNREGISTER(2)
WRITE	WRITE(8)
WRITE_FLUSH	WRITE_FLUSH(9)
YIELD	YIELD(12)

See also

- "StreamErrorContext property" on page 386

ULStreamErrorID enumeration

UL Ext.: Enumerates the network layers that may report errors during synchronization.

Prototypes

' Visual Basic

```
Public Enum ULStreamErrorID
  Inherits Short
```

// C#

```
public enum ULStreamErrorID :
  short
```

Member name

Member name	Description
ACTIVESYNC	ACTIVESYNC(22)
CERTICOM	CERTICOM(11)
CERTICOM_SSL	CERTICOM_SSL(13)
CERTICOM_TLS	CERTICOM_TLS(14)
DH_CAST	DH_CAST(9)
EMAIL	EMAIL(20)
FAKE	FAKE(2)
FILE	FILE(21)
HTTP	HTTP(7)
HTTPS	HTTPS(8)
JAVA_CERTICOM	JAVA_CERTICOM(12)
JAVA_RSA	JAVA_RSA(24)
NETTECH	NETTECH(5)
OPEN_SSL_RSA	OPEN_SSL_RSA(25)
PALM_CONDUIT	PALM_CONDUIT(3)
PALM_SS	PALM_SS(4)
PALM_SSL	PALM_SSL(26)
REPLAY	REPLAY(17)

Member name	Description
RIMBB	RIMBB(6)
RSA_TLS	RSA_TLS(23)
SECURE	SECURE(10)
SERIAL	SERIAL(1)
STRM	STRM(18)
TCPIP	TCPIP(0)
UDP	UDP(19)
WIRELESS	WIRELESS(16)
WIRESTRM	WIRESTRM(15)

See also

- “StreamErrorID property” on page 386

ULStreamType enumeration

UL Ext.: Enumerates the types of MobiLink synchronization streams to use for synchronization.

Prototypes

' Visual Basic

```
Public Enum ULStreamType
  Inherits Short
```

// C#

```
public enum ULStreamType :
  short
```

Remarks

For information on configuring specific stream types, see "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*].

Member name

Member name	Description
HTTP	Synchronize via HTTP. The HTTP stream uses TCP/IP as its underlying transport. UltraLite applications act as Web browsers and the MobiLink server acts as a Web server. UltraLite applications send POST requests to send data to the server and GET requests to read data from the server.
HTTPS	Synchronize via HTTPS (HTTP with transport-layer security). **Separately licensed component required** ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations. See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].
TCPIP	Synchronize via TCP/IP.
TLS	Synchronize via TCP/IP with transport layer security. **Separately licensed component required** ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations. See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].

See also

- "Stream property" on page 364

ULSyncParms class

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

Prototypes

' Visual Basic

NotInheritable Public 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 SyncParms property.

At most, only one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode enumeration SQLException is thrown by Synchronize method.

Other sources of ULSQLCode enumeration errors include not specifying a Stream property value or a Version property value.

See also

- "ULSyncParms members" on page 356
- "ULConnection class" on page 81
- "SyncParms property" on page 91
- "Synchronize method" on page 107

ULSyncParms members

Public properties

Member name	Description
AuthenticationParms property	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
CheckpointStore property	Specifies whether the client should perform extra store checkpoints to control the growth of the database store during synchronization.
DisableConcurrency property	Specifies whether concurrent access to UltraLite while performing a synchronization.
DownloadOnly property	Specifies whether to disable or enable uploads when synchronizing.

Member name	Description
KeepPartialDownload property	Specifies whether to disable or enable partial downloads when synchronizing.
NewPassword property	Specifies a new MobiLink password for the user specified with UserName.
Password property	The MobiLink password for the user specified by UserName.
PingOnly property	Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.
PublicationMask property	Specifies the publications to be synchronized.
ResumePartialDownload property	Specifies whether to resume or discard a previous partial download.
SendColumnNames property	Specifies whether the client should send column names to the MobiLink server during synchronization.
SendDownloadAck property	Specifies whether the client should send a download acknowledgement to the MobiLink server during synchronization. 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	Specifies the MobiLink synchronization stream to use for synchronization.
StreamParms property	Specifies the parameters to configure the synchronization stream.
TableOrder property	Specifies the order tables should be uploaded to the consolidated database.
UploadOnly property	Specifies whether to disable or enable downloads when synchronizing.
UserName property	The user name that uniquely identifies the MobiLink client to the MobiLink server.
Version property	Specifies which synchronization script to use.

Public methods

Member name	Description
CopyFrom method	Copies the properties of the specified ULSyncParms class object to this ULSyncParms class object.

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Prototypes

' 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).

CheckpointStore property

Specifies whether the client should perform extra store checkpoints to control the growth of the database store during synchronization.

Prototypes

' Visual Basic

Public Property **CheckpointStore** As Boolean

// C#

public bool **CheckpointStore** {get;set;}

Property value

True to specify that the client should perform extra store checkpoints. The default is false, meaning only required checkpointing is done.

Remarks

The checkpoint operation adds I/O operations for the application, and so slows synchronization. This option is most useful for large downloads with many updates. Devices with slow flash memory might not want to pay the performance penalty associated with additional checkpoints.

DisableConcurrency property

Specifies whether concurrent access to UltraLite while performing a synchronization.

Prototypes

' Visual Basic

Public Property **DisableConcurrency** As Boolean

// C#

public bool **DisableConcurrency** {get;set;}

Property value

True to disable concurrent access to UltraLite while performing a synchronization, false to enable concurrent access. The default is false.

Remarks

By default, other threads can perform UltraLite operations while a thread is synchronizing. When concurrent synchronization is disabled, other threads block on UltraLite calls until synchronization completes.

DownloadOnly property

Specifies whether to disable or enable uploads when synchronizing.

Prototypes

' 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 (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode enumeration SQLException is thrown by Synchronize method.

See also

- "ULSyncParms class" on page 356
- "ULSyncParms members" on page 356
- "UploadOnly property" on page 366

KeepPartialDownload property

Specifies whether to disable or enable partial downloads when synchronizing.

Prototypes

' 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.PartialDownloadRetained property 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.ResumePartialDownload property 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 PublicationMask 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 property and UserName property. The fields KeepPartialDownload and DisableConcurrency property 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 RollbackPartialDownload method 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 356

- "ULSyncParms members" on page 356
- "PartialDownloadRetained property" on page 385
- "ResumePartialDownload property" on page 363
- "RollbackPartialDownload method" on page 106

NewPassword property

Specifies a new MobiLink password for the user specified with UserName.

Prototypes

' 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 356
- "ULSyncParms members" on page 356
- "UserName property" on page 367

Password property

The MobiLink password for the user specified by UserName.

Prototypes

' 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 356
- "ULSyncParms members" on page 356
- "NewPassword property" on page 361
- "UserName property" on page 367

PingOnly property

Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.

Prototypes

' 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 (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode enumeration SQLException is thrown by Synchronize method.

PublicationMask property

Specifies the publications to be synchronized.

Prototypes

' Visual Basic

Public Property **PublicationMask** As Integer

// C#

public int **PublicationMask** {get;set;}

Property value

A bitwise combination of publication masks, the special value SYNC_ALL_PUBS field, or the special value SYNC_ALL_DB field. The default is SYNC_ALL_DB field. For more information on publication masks, see ULPublicationSchema class.

See also

- "ULSyncParms class" on page 356
- "ULSyncParms members" on page 356
- "Mask property" on page 288

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Prototypes

' 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 (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode enumeration SQLException is thrown by Synchronize method.

For more information on partial downloads, see KeepPartialDownload property.

See also

- "ULSyncParms class" on page 356
- "ULSyncParms members" on page 356
- "PartialDownloadRetained property" on page 385

SendColumnNames property

Specifies whether the client should send column names to the MobiLink server during synchronization.

Prototypes

' 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.

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).

Prototypes

' 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.

Stream property

Specifies the MobiLink synchronization stream to use for synchronization.

Prototypes

' Visual Basic

Public Property **Stream** As ULStreamType

// C#

public ULStreamType **Stream** {get;set;}

Property value

One of the ULStreamType enumeration values specifying the type of synchronization stream to use. The default is ULStreamType enumeration.

Remarks

Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. These parameters are supplied by the StreamParms property.

If the stream type is set to a value that is invalid for the platform, the stream type is set to ULStreamType enumeration.

See also

- "ULSyncParms class" on page 356
- "ULSyncParms members" on page 356
- "ULStreamType enumeration" on page 354
- "StreamParms property" on page 365

StreamParms property

Specifies the parameters to configure the synchronization stream.

Prototypes

' 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 on configuring specific stream types, see "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*].

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 356
- "ULSyncParms members" on page 356

- "Stream property" on page 364
- "ULStreamType enumeration" on page 354

TableOrder property

Specifies the order tables should be uploaded to the consolidated database.

Prototypes

' Visual Basic

Public Property **TableOrder** As String

// C#

public string **TableOrder** {get;set;}

Property value

A string, in the form of a comma-separated list of table names. Tables names can be quoted using either single or double quotes. The default is a null reference (Nothing in Visual Basic), which does not override the default ordering of tables.

Remarks

If the foreign keys on your consolidated database match the foreign keys on your remote UltraLite database and there are no foreign key cycles, then you most likely don't need to use this feature. However, if you have tables that are part of foreign key cycles then list all tables that are part of a cycle in the TableOrder field. If you have tables with different foreign key relationships on the consolidated then also list these tables in the TableOrder field.

All tables that you don't specify will be appropriately sorted based of the foreign keys defined in the remote database.

UploadOnly property

Specifies whether to disable or enable downloads when synchronizing.

Prototypes

' 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 (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode enumeration SQLException is thrown by Synchronize method.

See also

- "ULSyncParms class" on page 356
- "ULSyncParms members" on page 356
- "DownloadOnly property" on page 359

UserName property

The user name that uniquely identifies the MobiLink client to the MobiLink server.

Prototypes

' 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 356
- "ULSyncParms members" on page 356
- "Password property" on page 361

Version property

Specifies which synchronization script to use.

Prototypes

' 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.

CopyFrom method

Copies the properties of the specified ULSyncParms class object to this ULSyncParms class object.

Prototypes

```
' Visual Basic

Public Sub CopyFrom( _
   ByVal src As ULSyncParms _
)

// C#

public void CopyFrom(
   ULSyncParms src
);
```

Parameters

- **src** The object to copy from.

ULSyncProgressData class

UL Ext.: Returns synchronization progress monitoring data.

Prototypes

' Visual Basic

Public Class **ULSyncProgressData**

// C#

public class **ULSyncProgressData**

See also

- "ULSyncProgressData members" on page 369
- "ULSyncProgressListener interface" on page 379

ULSyncProgressData members

Public static fields (shared)

Member name	Description
FLAG_IS_BLOCKING field	A flag indicating that the synchronization is blocked awaiting a response from the MobiLink server.

Public properties

Member name	Description
ErrorMessage property	Returns the error message describing the error that occurred during synchronization.
Flags property	Returns the current synchronization flags indicating additional information relating to the current state.
ReceivedBytes property	Returns the number of bytes received so far. This information is updated for all states.
ReceivedDeletes property	Returns the number of deleted rows received so far.
ReceivedInserts property	Returns the number of inserted rows received so far.
ReceivedUpdates property	Returns the number of updated rows received so far.
SentBytes property	Returns the number of bytes sent so far. This information is updated for all states.
SentDeletes property	Returns the number of deleted rows sent so far.
SentInserts property	Returns the number of inserted rows sent so far.

Member name	Description
SentUpdates property	Returns the number of updated rows sent so far.
SQLCode property	Returns the SQL code for synchronization.
State property	Returns the current synchronization state.
SyncParms property	Returns a reference to the connection's SyncParms object.
SyncResult property	Returns a reference to the connection's SyncResult object. This object is only updated with ULSyncProgressState enumeration and UL-SyncProgressState enumeration events.
TableCount property	A count of the tables sent or received (TableCount of TableTotal) so far.
TableIndex property	Returns the index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).
TableTotal property	Returns the number of tables being synchronized.

FLAG_IS_BLOCKING field

A flag indicating that the synchronization is blocked awaiting a response from the MobiLink server.

Prototypes

' Visual Basic

```
Public Shared FLAG_IS_BLOCKING As Integer
```

// C#

```
public const int FLAG_IS_BLOCKING;
```

ErrorMessage property

Returns the error message describing the error that occurred during synchronization.

Prototypes

' Visual Basic

```
Public Readonly Property ErrorMessage As String
```

// C#

```
public string  ErrorMessage {get;}
```

Property value

A string describing the error that occurred during synchronization.

See also

- "ULSyncProgressData class" on page 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381

Flags property

Returns the current synchronization flags indicating additional information relating to the current state.

Prototypes

' 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 369
- "ULSyncProgressData members" on page 369
- "FLAG_IS_BLOCKING field" on page 370

ReceivedBytes property

Returns the number of bytes received so far. This information is updated for all states.

Prototypes

' 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 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381

ReceivedDeletes property

Returns the number of deleted rows received so far.

Prototypes

' 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 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381

ReceivedInserts property

Returns the number of inserted rows received so far.

Prototypes

' 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 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381

ReceivedUpdates property

Returns the number of updated rows received so far.

Prototypes

' 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 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381

SentBytes property

Returns the number of bytes sent so far. This information is updated for all states.

Prototypes

' 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 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381

SentDeletes property

Returns the number of deleted rows sent so far.

Prototypes

' 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 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381

SentInserts property

Returns the number of inserted rows sent so far.

Prototypes

' 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 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381

SentUpdates property

Returns the number of updated rows sent so far.

Prototypes

' 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 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381

SQLCode property

Returns the SQL code for synchronization.

Prototypes

' Visual Basic

Public Readonly Property **SQLCode** As ULSQLCode

// C#

public ULSQLCode **SQLCode** {get;}

Property value

The ULSQLCode enumeration value for any synchronization error.

See also

- "ULSyncProgressData class" on page 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381

State property

Returns the current synchronization state.

Prototypes

' Visual Basic

Public Readonly Property **State** As ULSyncProgressState

// C#

public ULSyncProgressState **State** {get;}

Property value

One of the ULSyncProgressState enumeration values specifying the current synchronization state.

See also

- "ULSyncProgressData class" on page 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381

SyncParms property

Returns a reference to the connection's SyncParms object.

Prototypes

' Visual Basic

Public Readonly Property **SyncParms** As ULSyncParms

// C#

public ULSyncParms **SyncParms** {get;}

Property value

A reference to the SyncParms property object.

See also

- "ULSyncProgressData class" on page 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381
- "SyncParms property" on page 91

SyncResult property

Returns a reference to the connection's SyncResult object. This object is only updated with ULSyncProgressState enumeration and ULSyncProgressState enumeration events.

Prototypes

' Visual Basic

Public Readonly Property **SyncResult** As ULSyncResult

// C#

public ULSyncResult **SyncResult** {get;}

Property value

A reference to the SyncResult property object.

See also

- "ULSyncProgressData class" on page 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381

TableCount property

A count of the tables sent or received (TableCount of TableTotal) so far.

Prototypes

' Visual Basic

Public Readonly Property **TableCount** As Integer

// C#

public int **TableCount** {get;}

Property value

The count of tables sent or received.

See also

- "ULSyncProgressData class" on page 369
- "ULSyncProgressData members" on page 369
- "ULSyncProgressState enumeration" on page 381
- "ULSyncProgressState enumeration" on page 381

TableIndex property

Returns the index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).

Prototypes

' Visual Basic

Public Readonly Property **TableIndex** As Integer

// C#

public int **TableIndex** {get;}

Property value

The index of the table currently being synchronized. Tables are numbered 1 to TableCount property.

See also

- “ULSyncProgressData class” on page 369
- “ULSyncProgressData members” on page 369
- “ULSyncProgressState enumeration” on page 381
- “ULSyncProgressState enumeration” on page 381

TableTotal property

Returns the number of tables being synchronized.

Prototypes

' Visual Basic

Public Readonly Property **TableTotal** As Integer

// C#

public int **TableTotal** {get;}

Property value

The number of tables being synchronized.

See also

- “ULSyncProgressData class” on page 369
- “ULSyncProgressData members” on page 369
- “ULSyncProgressState enumeration” on page 381
- “ULSyncProgressState enumeration” on page 381

ULSyncProgressListener interface

UL Ext.: The listener interface for receiving synchronization progress events.

Prototypes

' Visual Basic

Public Interface **ULSyncProgressListener**

// C#

public interface **ULSyncProgressListener**

See also

- "ULSyncProgressListener members" on page 379
- "Synchronize method" on page 108

ULSyncProgressListener members

Public methods

Member name	Description
SyncProgressed method	Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue.

SyncProgressed method

Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue.

Prototypes

' Visual Basic

```
Public Function SyncProgressed( _
  ByVal data As ULSyncProgressData _
) As Boolean
```

// C#

```
public bool SyncProgressed(
  ULSyncProgressData data
);
```

Parameters

- **data** A ULSyncProgressData class 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.

ULSyncProgressState enumeration

UL Ext.: Enumerates all the states that can occur while synchronizing.

Prototypes

' Visual Basic

```
Public Enum ULSyncProgressState
  Inherits Short
```

// C#

```
public enum ULSyncProgressState :
  short
```

Member name

Member name	Description
STATE_CANCELLED	Synchronization has been cancelled.
STATE_COMMITTING_DOWNLOAD	The download is being committed. The final count of rows received is included with this event. See ReceivedBytes property, ReceivedInserts property, ReceivedUpdates property, and ReceivedDeletes property.
STATE_CONNECTING	The synchronization stream has been built, but is not yet opened.
STATE_DISCONNECTING	The synchronization stream is about to be closed.
STATE_DONE	Synchronization has successfully completed. The connection's SyncResult property object has been updated.
STATE_ERROR	Synchronization has completed, but an error occurred. Check SyncResult property, ErrorMessage property, and SQLCode property for details.
STATE_FINISHING_UPLOAD	The upload is completing. The final count of rows sent is included with this event. See SentBytes property, SentInserts property, SentUpdates property, and SentDeletes property.
STATE_LAST	The last state entered by synchronization. This state is always entered and always after any other state. Other ULSyncProgressData fields may not contain valid data.
STATE_RECEIVING_DATA	Data for the current table is being received. ReceivedBytes property, ReceivedInserts property, ReceivedUpdates property, and ReceivedDeletes property have been updated.
STATE_RECEIVING_TABLE	A table is being received. Progress can be monitored using TableIndex property and TableCount property.
STATE_RECEIVING_UPLOAD_ACK	An acknowledgement that the upload is complete is being received.

Member name	Description
STATE_ROLLING_BACK_DOWNLOAD	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.
STATE_SENDING_DATA	Data for the current table is being sent. SentBytes property, SentInserts property, SentUpdates property, and SentDeletes property have been updated.
STATE_SENDING_DOWNLOAD_ACK	An acknowledgement that the download is complete is being sent.
STATE_SENDING_HEADER	The synchronization stream has been opened and the header is about to be sent.
STATE_SENDING_TABLE	A table is being sent. Progress can be monitored using TableIndex property and TableCount property.
STATE_STARTING	No synchronization actions have been taken yet.

See also

- "ULSyncProgressData class" on page 369

ULSyncResult class

UL Ext.: Represents the status of the last synchronization.

Prototypes

' 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 SyncResult property. A ULSyncResult instance is only valid while that connection is open.

See also

- "ULSyncResult members" on page 383
- "SyncResult property" on page 92
- "Synchronize method" on page 107

ULSyncResult members

Public properties

Member name	Description
AuthStatus property	Returns the authorization status code for the last synchronization attempt.
AuthValue property	Returns the return value from custom user authentication synchronization scripts.
IgnoredRows property	Checks whether any uploaded rows were ignored during the last synchronization.
PartialDownloadRetained property	Checks whether a partial download was retained during the last synchronization.
StreamErrorCode property	Returns the error reported by the stream itself.
StreamErrorContext property	Returns the basic network operation being performed when the stream error occurred.
StreamErrorID property	Returns the ID of the network layer reporting an error.
StreamErrorSystem property	Returns the stream error system-specific code.
Timestamp property	Returns the timestamp of the last synchronization.

Member name	Description
UploadOK property	Checks whether the last upload synchronization was successful.

Protected methods

Member name	Description
Finalize method	Releases unmanaged resources and performs other cleanup operations before the ULSyncResult is reclaimed by garbage collection.

AuthStatus property

Returns the authorization status code for the last synchronization attempt.

Prototypes

' Visual Basic

Public Readonly Property **AuthStatus** As ULAuthStatusCode

// C#

public ULAuthStatusCode **AuthStatus** {get;}

Property value

One of the ULAuthStatusCode enumeration values denoting the authorization status for the last synchronization attempt.

AuthValue property

Returns the return value from custom user authentication synchronization scripts.

Prototypes

' 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.

IgnoredRows property

Checks whether any uploaded rows were ignored during the last synchronization.

Prototypes

' 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 383
- "ULSyncResult members" on page 383
- "DownloadOnly property" on page 359

PartialDownloadRetained property

Checks whether a partial download was retained during the last synchronization.

Prototypes

' 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 383
- "ULSyncResult members" on page 383
- "KeepPartialDownload property" on page 360

StreamErrorCode property

Returns the error reported by the stream itself.

Prototypes

' Visual Basic

Public Readonly Property **StreamErrorCode** As ULStreamErrorCode

// C#

public ULStreamErrorCode **StreamErrorCode** {get;}

Property value

One of the ULStreamErrorCode enumeration values denoting the error reported by the stream itself, ULStreamErrorCode enumeration if no error occurred.

StreamErrorContext property

Returns the basic network operation being performed when the stream error occurred.

Prototypes

' Visual Basic

Public Readonly Property **StreamErrorContext** As ULStreamErrorContext

// C#

public ULStreamErrorContext **StreamErrorContext** {get;}

Property value

One of the ULStreamErrorContext enumeration values denoting which basic network operation was being performed when the stream error occurred.

StreamErrorID property

Returns the ID of the network layer reporting an error.

Prototypes

' Visual Basic

Public Readonly Property **StreamErrorID** As ULStreamErrorID

// C#

public ULStreamErrorID **StreamErrorID** {get;}

Property value

One of the ULStreamErrorID enumeration values denoting the ID of the network layer reporting an error.

StreamErrorSystem property

Returns the stream error system-specific code.

Prototypes

' Visual Basic

Public Readonly Property **StreamErrorSystem** As Integer

// C#

public int **StreamErrorSystem** {get;}

Property value

An integer denoting the stream error system-specific code.

Timestamp property

Returns the timestamp of the last synchronization.

Prototypes

' Visual Basic

Public Readonly Property **Timestamp** As Date

// C#

public DateTime **Timestamp** {get;}

Property value

A DateTime specifying the timestamp of the last synchronization.

UploadOK property

Checks whether the last upload synchronization was successful.

Prototypes

' 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.

Finalize method

Releases unmanaged resources and performs other cleanup operations before the ULSyncResult is reclaimed by garbage collection.

Prototypes

' Visual Basic

Overrides Protected Sub **Finalize()**

// C#

protected override void **Finalize();**

ULTable class

UL Ext.: Represents a table in an UltraLite database.

Prototypes

' 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 ExecuteTable method of the ULCommand class.

Inherits: ULResultSet class

Implements: IDataReader, IDataRecord, IDisposable

ULTable members

Public properties

Member name	Description
Depth property (inherited from ULDataReader)	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
FieldCount property (inherited from ULDataReader)	Returns the number of columns in the cursor.
IsBOF property (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is before the first row.
IsClosed property (inherited from ULDataReader)	Checks whether the cursor is currently open.
IsEOF property (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is after the last row.
Item property (inherited from ULDataReader)	Returns the value of the specified column in its native format. In C#, this property is the indexer for the ULDataReader class.
Item property (inherited from ULDataReader)	Returns the value of the specified named column in its native format. In C#, this property is the indexer for the ULDataReader class.

Member name	Description
RecordsAffected property (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 (inherited from ULDataReader)	**UL Ext.:** Returns the number of rows in the cursor.
Schema property	Holds the table schema. This property is only valid while its connection is open.

Public methods

Member name	Description
AppendBytes method (inherited from ULResultSet)	Appends the specified subset of the specified array of Bytes to the new value for the specified ULDbType enumeration column.
AppendChars method (inherited from ULResultSet)	Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType enumeration column.
Close method (inherited from ULDataReader)	Closes the cursor.
Delete method (inherited from ULResultSet)	Deletes the current row.
DeleteAllRows method	Deletes all rows in the table.
Dispose method (inherited from ULDataReader)	Releases the unmanaged resources used by the ULDataReader and optionally releases the managed resources.
FindBegin method	Prepares to perform a new Find on a table.
FindFirst method	Moves forward through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.
FindFirst method	Moves forward through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.
FindLast method	Moves backward through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.
FindLast method	Moves backward through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.
FindNext method	Continues a FindFirst method 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.

Member name	Description
FindNext method	Continues a FindFirst method search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or partial set of values in the current index.
FindPrevious method	Continues a FindLast method 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 FindLast method search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or partial set of values in the current index.
GetBoolean method (inherited from ULDataReader)	Returns the value for the specified column as a Boolean.
GetByte method (inherited from ULDataReader)	Returns the value for the specified column as an unsigned 8-bit value (Byte).
GetBytes method (inherited from ULDataReader)	Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination Byte array.
GetBytes method (inherited from ULDataReader)	**UL Ext.:** Returns the value for the specified column as an array of Bytes. Only valid for columns of type ULDbType enumeration, ULDbType enumeration, or ULDbType enumeration.
GetChar method (inherited from ULDataReader)	This method is not supported in UltraLite.NET.
GetChars method (inherited from ULDataReader)	Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination System.Char array.
GetData method (inherited from ULDataReader)	This method is not supported in UltraLite.NET.
GetDataTypeName method (inherited from ULDataReader)	Returns the name of the specified column's provider data type.
GetDateTime method (inherited from ULDataReader)	Returns the value for the specified column as a DateTime with millisecond accuracy.
GetDecimal method (inherited from ULDataReader)	Returns the value for the specified column as a Decimal.
GetDouble method (inherited from ULDataReader)	Returns the value for the specified column as a Double.
GetFieldType method (inherited from ULDataReader)	Returns the Type most appropriate for the specified column.
GetFloat method (inherited from ULDataReader)	Returns the value for the specified column as a Single.

Member name	Description
GetGuid method (inherited from ULDataReader)	Returns the value for the specified column as a UUID (Guid).
GetInt16 method (inherited from ULDataReader)	Returns the value for the specified column as an Int16.
GetInt32 method (inherited from ULDataReader)	Returns the value for the specified column as an Int32.
GetInt64 method (inherited from ULDataReader)	Returns the value for the specified column as an Int64.
GetName method (inherited from ULDataReader)	Returns the name of the specified column.
GetOrdinal method (inherited from ULDataReader)	Returns the column ID of the named column.
GetSchemaTable method (inherited from ULDataReader)	Returns a DataTable that describes the column metadata of the ULDataReader.
GetString method (inherited from ULDataReader)	Returns the value for the specified column as a String.
GetTimeSpan method (inherited from ULDataReader)	Returns the value for the specified column as a TimeSpan with millisecond accuracy.
GetUInt16 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt16.
GetUInt32 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt32.
GetUInt64 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt64.
GetValue method (inherited from ULDataReader)	Returns the value of the specified column in its native format.
GetValues method (inherited from ULDataReader)	Returns all the column values for the current row.
Insert method	Inserts a new row with the current column values (specified using the set methods). Each insert must be preceded by a call to InsertBegin method.
InsertBegin method	Prepares to insert a new row into the table by setting all current column values to their default values.
IsDBNull method (inherited from ULDataReader)	Checks whether the value from the specified column is NULL.

Member name	Description
LookupBackward method	Moves backward through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.
LookupBackward method	Moves backward through the table from the beginning, looking for a row that matches or is less than a value or partial set of values in the current index.
LookupBegin method	Prepares to perform a new lookup on 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 method	Moves forward through the table from the beginning, looking for a row that matches or is greater than a value or full set of values in the current index.
LookupForward method	Moves forward through the table from the beginning, looking for a row that matches or is greater than a value or partial set of values in the current index.
MoveAfterLast method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to after the last row of the cursor.
MoveBeforeFirst method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to before the first row of the cursor.
MoveFirst method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the first row of the cursor.
MoveLast method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the last row of the cursor.
MoveNext method (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 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the previous row or before the first row.
MoveRelative method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor relative to the current row.
NextResult method (inherited from ULDataReader)	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
Read method (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 (inherited from ULResultSet)	Sets the value for the specified column using a Boolean.
SetByte method (inherited from ULResultSet)	Sets the value for the specified column using a Byte (unsigned 8-bit integer).

Member name	Description
SetBytes method (inherited from ULResultSet)	Sets the value for the specified column using an array of Bytes.
SetDateTime method (inherited from ULResultSet)	Sets the value for the specified column using a DateTime.
SetDBNull method (inherited from ULResultSet)	Sets a column to NULL.
SetDecimal method (inherited from ULResultSet)	Sets the value for the specified column using a Decimal.
SetDouble method (inherited from ULResultSet)	Sets the value for the specified column using a Double.
SetFloat method (inherited from ULResultSet)	Sets the value for the specified column using a Single.
SetGuid method (inherited from ULResultSet)	Sets the value for the specified column using a Guid.
SetInt16 method (inherited from ULResultSet)	Sets the value for the specified column using an Int16.
SetInt32 method (inherited from ULResultSet)	Sets the value for the specified column using an Int32.
SetInt64 method (inherited from ULResultSet)	Sets the value for the specified column using an Int64.
SetString method (inherited from ULResultSet)	Sets the value for the specified column using a String.
SetTimeSpan method (inherited from ULResultSet)	Sets the value for the specified column using a TimeSpan.
SetToDefault method (inherited from ULResultSet)	Sets the value for the specified column to its default value.
SetUInt16 method (inherited from ULResultSet)	Sets the value for the specified column using a UInt16.
SetUInt32 method (inherited from ULResultSet)	Sets the value for the specified column using an UInt32.
SetUInt64 method (inherited from ULResultSet)	Sets the value for the specified column using a UInt64.
Truncate method	Deletes all rows in the table while temporarily activating a stop synchronization delete.
Update method (inherited from ULResultSet)	Updates the current row with the current column values (specified using the set methods).

Member name	Description
UpdateBegin method	Prepares to update the current row in the table.

Protected methods

Member name	Description
Finalize method	Releases unmanaged resources and performs other cleanup operations before the ULTable is reclaimed by garbage collection.

Schema property

Holds the table schema. This property is only valid while its connection is open.

Prototypes

' Visual Basic

Public Readonly Property **Schema** As ULTableSchema

// C#

public ULTableSchema **Schema** {get;}

Property value

The ULTableSchema class 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 GetSchemaTable method.

DeleteAllRows method

Deletes all rows in the table.

Prototypes

' 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 StopSynchronizationDelete method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "Truncate method" on page 408

Finalize method

Releases unmanaged resources and performs other cleanup operations before the ULTable is reclaimed by garbage collection.

Prototypes

' Visual Basic

Overrides Protected Sub **Finalize()**

// C#

protected override void **Finalize();**

FindBegin method

Prepares to perform a new Find on a table.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "FindFirst method" on page 397
- "FindFirst method" on page 398
- "FindLast method" on page 398
- "FindLast method" on page 399

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.

Prototypes

' Visual Basic

Overloads 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 (IsEOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "FindBegin method" on page 396
- "FindNext method" on page 400
- "FindPrevious method" on page 402
- "FindFirst method" on page 398

FindFirst 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.

Prototypes

' Visual Basic

```
Overloads 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 (IsEOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "FindBegin method" on page 396
- "FindNext method" on page 401
- "FindPrevious method" on page 402
- "FindFirst method" on page 397

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.

Prototypes

' Visual Basic

Overloads 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 (IsBOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULTable class” on page 389
- “ULTable members” on page 389
- “FindBegin method” on page 396
- “FindNext method” on page 400
- “FindPrevious method” on page 402
- “FindLast method” on page 399

FindLast 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.

Prototypes

' Visual Basic

```
Overloads 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 (IsBOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "FindBegin method" on page 396
- "FindNext method" on page 401
- "FindPrevious method" on page 402
- "FindLast method" on page 398

FindNext method

Continues a FindFirst method 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.

Prototypes

' Visual Basic

Overloads 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 (IsEOF property).

FindNext behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "FindFirst method" on page 397
- "FindNext method" on page 401

FindNext method

Continues a FindFirst method 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.

Prototypes

' Visual Basic

```
Overloads 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 (IsEOF property).

FindNext behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389

- "ULTable members" on page 389
- "FindFirst method" on page 398
- "FindNext method" on page 400

FindPrevious method

Continues a FindLast method 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.

Prototypes

' Visual Basic

Overloads 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 (IsBOF property).

FindPrevious behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "FindLast method" on page 398
- "FindPrevious method" on page 402

FindPrevious method

Continues a FindLast method 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.

Prototypes

' Visual Basic

Overloads 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 (IsBOF property).

FindPrevious behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "FindLast method" on page 399
- "FindPrevious method" on page 402

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 InsertBegin method.

Prototypes

' Visual Basic

```
Public Sub Insert()
```

// C#

```
public void Insert();
```

Exceptions

- ULException class - A SQL error occurred.

InsertBegin method

Prepares to insert a new row into the table by setting all current column values to their default values.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "Insert method" on page 403

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.

Prototypes

' Visual Basic

Overloads 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 (IsBOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "LookupBegin method" on page 406
- "LookupBackward method" on page 405

LookupBackward 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.

Prototypes

' Visual Basic

```
Overloads 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 (IsBOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "LookupBegin method" on page 406
- "LookupBackward method" on page 404

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.

Prototypes

' Visual Basic

Public Sub **LookupBegin()**

// C#

public void **LookupBegin();**

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "LookupForward method" on page 406
- "LookupForward method" on page 407
- "LookupBackward method" on page 404
- "LookupBackward method" on page 405

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.

Prototypes

' Visual Basic

Overloads 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 (IsEOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "LookupBegin method" on page 406
- "LookupForward method" on page 407

LookupForward 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.

Prototypes

' Visual Basic

```
Overloads 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 (IsEOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "LookupBegin method" on page 406
- "LookupForward method" on page 406

Truncate method

Deletes all rows in the table while temporarily activating a stop synchronization delete.

Prototypes

' Visual Basic

Public Sub **Truncate()**

// C#

public void **Truncate();**

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 389
- "ULTable members" on page 389
- "DeleteAllRows method" on page 395

UpdateBegin method

Prepares to update the current row in the table.

Prototypes

' 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 Update method, 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.

Exceptions

- ULException class - A SQL error occurred.

ULTableSchema class

UL Ext.: Represents the schema of an UltraLite table.

Prototypes

' Visual Basic

```
NotInheritable Public Class ULTableSchema
  Inherits ULCursorSchema
```

// C#

```
public sealed class ULTableSchema :
  ULCursorSchema
```

Remarks

There is no constructor for this class. A ULTableSchema class object is attached to a table as its Schema property.

Inherits: ULCursorSchema class

ULTableSchema members

Public properties

Member name	Description
ColumnCount property (inherited from ULCursorSchema)	Returns the number of columns in the cursor.
IndexCount property	Returns the number of indexes on the table.
IsNeverSynchronized property	Checks whether the table is marked as never being synchronized.
IsOpen property (inherited from ULCursorSchema)	Checks whether the cursor schema is currently open.
Name property	Returns the name of the table.
PrimaryKey property	Returns the index schema of the primary key for the table.
UploadUnchangedRows property	Checks whether the database uploads rows that have not changed.

Public methods

Member name	Description
GetColumnDefaultValue method	Returns the default value of the specified column.

Member name	Description
GetColumnID method (inherited from ULCursorSchema)	Returns the column ID of the named column.
GetColumnName method (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
GetColumnPartitionSize method	Returns the global autoincrement partition size assigned to the specified column.
GetColumnPrecision method (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).
GetColumnScale method (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 (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).
GetColumnSQLName method (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
GetColumnULDbType method (inherited from ULCursorSchema)	Returns the UltraLite.NET data type of the column identified by the specified column ID.
GetIndex method	Returns the index schema of the named index.
GetIndexName method	Returns the name of the index identified by the specified index ID.
GetOptimalIndex method	The optimal index for searching a table using the specified column.
GetPublicationPredicate method	Returns the publication predicate for this table in the named publication.
GetSchemaTable method (inherited from ULCursorSchema)	Returns a DataTable that describes the column schema of the ULDataReader class.
IsColumnAutoIncrement method	Checks whether the specified column's default is set to autoincrement.
IsColumnCurrentDate method	Checks whether the specified column's default is set to the current date (ULDbType enumeration).
IsColumnCurrentTime method	Checks whether the specified column's default is set to the current time (ULDbType enumeration).
IsColumnCurrentTimestamp method	Checks whether the specified column's default is set to the current timestamp (ULDbType enumeration).
IsColumnGlobalAutoIncrement method	Checks whether the specified column's default is set to global autoincrement.

Member name	Description
IsColumnNewUUID method	Checks whether the specified column's default is set to a new UUID (Guid).
IsColumnNullable method	Checks whether the specified column is nullable.
IsInPublication method	Checks whether the table is contained in the named publication.

IndexCount property

Returns the number of indexes on the table.

Prototypes

' 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.

IsNeverSynchronized property

Checks whether the table is marked as never being synchronized.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

Name property

Returns the name of the table.

Prototypes

' Visual Basic

```
Public Readonly Property Name As String
```

// C#

```
public string  Name {get;}
```

Property value

The name of the table as a string.

Exceptions

- ULException class - A SQL error occurred.

PrimaryKey property

Returns the index schema of the primary key for the table.

Prototypes

' Visual Basic

```
Public Readonly Property PrimaryKey As ULIndexSchema
```

// C#

```
public ULIndexSchema PrimaryKey {get;}
```

Property value

A ULIndexSchema class object representing the primary key for the table.

Exceptions

- ULException class - A SQL error occurred.

UploadUnchangedRows property

Checks whether the database uploads rows that have not changed.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

GetColumnDefaultValue method

Returns the default value of the specified column.

Prototypes

' 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,ColumnCount property-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.

Exceptions

- ULException class - A SQL error occurred.

GetColumnPartitionSize method

Returns the global autoincrement partition size assigned to the specified column.

Prototypes

' 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,ColumnCount property-1]. The first column in the table has an ID value of zero.

Return value

The column's global autoincrement partition size as a UInt64.

Remarks

All global autoincrement columns in a given table share the same global autoincrement partition.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULTableSchema class” on page 410
- “ULTableSchema members” on page 410
- “IsColumnGlobalAutoIncrement method” on page 420

GetIndex method

Returns the index schema of the named index.

Prototypes

' 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 class object representing the named index.

Exceptions

- ULException class - A SQL error occurred.

GetIndexName method

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

Prototypes

```
' 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 property].

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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTableSchema class" on page 410
- "ULTableSchema members" on page 410
- "IndexCount property" on page 412

GetOptimalIndex method

The optimal index for searching a table using the specified column.

Prototypes

' 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,ColumnCount property-1]. The first column in the table has an ID value of zero.

Return value

A ULIndexSchema class 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.

Exceptions

- ULException class - A SQL error occurred.

GetPublicationPredicate method

Returns the publication predicate for this table in the named publication.

Prototypes

' 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.

Exceptions

- ULException class - A SQL error occurred.

IsColumnAutoIncrement method

Checks whether the specified column's default is set to autoincrement.

Prototypes

' 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,ColumnCount property-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.

Exceptions

- ULException class - A SQL error occurred.

IsColumnCurrentDate method

Checks whether the specified column's default is set to the current date (ULDbType enumeration).

Prototypes

' 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,ColumnCount property-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.

Exceptions

- ULException class - A SQL error occurred.

IsColumnCurrentTime method

Checks whether the specified column's default is set to the current time (ULDbType enumeration).

Prototypes

```
' 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,ColumnCount property-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.

Exceptions

- ULException class - A SQL error occurred.

IsColumnCurrentTimestamp method

Checks whether the specified column's default is set to the current timestamp (ULDbType enumeration).

Prototypes

' 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,ColumnCount property-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.

Exceptions

- ULException class - A SQL error occurred.

IsColumnGlobalAutoIncrement method

Checks whether the specified column's default is set to global autoincrement.

Prototypes

' 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,ColumnCount property-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.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTableSchema class" on page 410
- "ULTableSchema members" on page 410
- "GetColumnPartitionSize method" on page 415
- "DatabaseID property" on page 88

IsColumnNewUUID method

Checks whether the specified column's default is set to a new UUID (Guid).

Prototypes

' 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,ColumnCount property-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.

Exceptions

- ULException class - A SQL error occurred.

IsColumnNullable method

Checks whether the specified column is nullable.

Prototypes

' 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,ColumnCount property-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.

Exceptions

- ULException class - A SQL error occurred.

IsInPublication method

Checks whether the table is contained in the named publication.

Prototypes

```
' 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.

Exceptions

- ULException class - A SQL error occurred.

ULTransaction class

Represents a SQL transaction.

Prototypes

' Visual Basic

NotInheritable Public Class **ULTransaction**

// C#

public sealed class **ULTransaction**

Remarks

There is no constructor for ULTransaction. To obtain a ULTransaction object, use the BeginTransaction method. To associate a command with a transaction, use the Transaction property.

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.

Implements: IDbTransaction, IDisposable

ULTransaction members

Public properties

Member name	Description
Connection property	Returns the connection associated with the transaction.
IsolationLevel property	Returns the isolation level for the transaction.

Public methods

Member name	Description
Commit method	Commits the database transaction.
Dispose method	Releases the resources used by the ULTransaction and rolls back any uncommitted commands.
Rollback method	Rolls back the transaction's outstanding changes to the database.

Connection property

Returns the connection associated with the transaction.

Prototypes

' Visual Basic

```
Public Readonly Property Connection As ULConnection
```

// C#

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

Property value

The ULConnection class 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 IDbTransaction.Connection.

See also

- "ULTransaction class" on page 423
- "ULTransaction members" on page 423
- "BeginTransaction method" on page 92

IsolationLevel property

Returns the isolation level for the transaction.

Prototypes

' Visual Basic

```
NotOverridable Public Readonly Property IsolationLevel As IsolationLevel _
  Implements IDbTransaction.IsolationLevel
```

// C#

```
public IsolationLevel IsolationLevel {get;}
```

Property value

One of the IsolationLevel values. UltraLite.NET only supports IsolationLevel.ReadUncommitted.

Implements

IDbTransaction.IsolationLevel

See also

- "ULTransaction class" on page 423
- "ULTransaction members" on page 423
- "BeginTransaction method" on page 92

Commit method

Commits the database transaction.

Prototypes

' Visual Basic

```
NotOverridable Public Sub Commit() _
  Implements IDbTransaction.Commit
```

// C#

```
public 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.

Implements

IDbTransaction.Commit

See also

- "ULTransaction class" on page 423
- "ULTransaction members" on page 423
- "Rollback method" on page 425

Dispose method

Releases the resources used by the ULTransaction and rolls back any uncommitted commands.

Prototypes

' Visual Basic

```
NotOverridable Public Sub Dispose() _
  Implements IDisposable.Dispose
```

// C#

```
public void Dispose();
```

Implements

IDisposable.Dispose

Rollback method

Rolls back the transaction's outstanding changes to the database.

Prototypes

' Visual Basic

```
NotOverridable Public Sub Rollback() _
  Implements IDbTransaction.Rollback
```

// C#

```
public 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.

Implements

IDbTransaction.Rollback

See also

- "ULTransaction class" on page 423
- "ULTransaction members" on page 423
- "Commit method" on page 425

CHAPTER 6

iAnywhere.Data.UltraLite namespace (.NET 2.0)

Contents

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 586 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 498.

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.

ULActiveSyncListener interface

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

Prototypes

Visual Basic
Public Interface **ULActiveSyncListener**

C#
public interface **ULActiveSyncListener**

See also

- "ULActiveSyncListener members" on page 430

ULActiveSyncListener members

Public methods

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

See also

- "ULActiveSyncListener interface" on page 430

ActiveSyncInvoked method

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

Prototypes

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 "SignalSyncIsComplete method" on page 591 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 430
- "ULActiveSyncListener members" on page 430

ULAuthStatusCode enumeration

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

Prototypes

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 825

ULBulkCopy class

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

Prototypes

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 434

ULBulkCopy members

Public constructors

Member name	Description
ULBulkCopy constructors	Initializes a ULBulkCopy object with the specified "ULConnection class" on page 498.

Public properties

Member name	Description
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.
BulkCopyTimeout property	Gets or sets the number of seconds for the operation to complete before it times out.
ColumnMappings property	Returns a collection of "ULBulkCopyColumnMapping class" on page 445 items. Column mappings define the relationships between columns in the data source and columns in the destination.
DestinationTableName property	Gets or sets the name of the destination table on the server.
NotifyAfter property	Specifies the number of rows to be processed before generating a notification event.

Public methods

Member name	Description
Close method	Closes the ULBulkCopy instance.
Dispose method	Disposes of the ULBulkCopy instance.
WriteToServer methods	Copies all rows in the supplied array of DataRow objects to a destination table specified by the "DestinationTableName property" on page 440 of the ULBulkCopy object.

Public events

Member name	Description
ULRowsCopied event	This event occurs every time the number of rows specified by the "NotifyAfter property" on page 440 have been processed.

See also

- "ULBulkCopy class" on page 434

ULBulkCopy constructors

Initializes a ULBulkCopy object with the specified "ULConnection class" on page 498.

ULBulkCopy(ULConnection) constructor

Prototypes

Visual Basic

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

C#

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

Parameters

- **connection** The already open "ULConnection class" on page 498 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 434

- "ULBulkCopy members" on page 434
- "ULBulkCopy constructors" on page 435

ULBulkCopy(String) constructor

Initializes a ULBulkCopy object with the specified connection string.

Prototypes

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 the "ConnectionString property" on page 503.

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 class" on page 531 object.

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

Implements: IDisposable

See also

- "ULBulkCopy class" on page 434
- "ULBulkCopy members" on page 434
- "ULBulkCopy constructors" on page 435

ULBulkCopy(String, ULBulkCopyOptions) constructor

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

Prototypes

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 the "ConnectionString property" on page 503.

- **copyOptions** A combination of values from the "ULBulkCopyOptions enumeration" on page 461 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 434
- "ULBulkCopy members" on page 434
- "ULBulkCopy constructors" on page 435

ULBulkCopy(ULConnection, ULBulkCopyOptions, ULTransaction) constructor

Initializes a ULBulkCopy object with the specified "ULConnection class" on page 498, copy options and "ULTransaction class" on page 862.

Prototypes

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 class" on page 498 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" on page 461 enumeration that determines how data source rows are copied to the destination table.
- **externalTransaction** An existing "ULTransaction class" on page 862 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 enumeration" on page 461 option.

Remarks

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

See also

- "ULBulkCopy class" on page 434
- "ULBulkCopy members" on page 434
- "ULBulkCopy constructors" on page 435

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.

Prototypes

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 434
- "ULBulkCopy members" on page 434

BulkCopyTimeout property

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

Prototypes

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 434
- "ULBulkCopy members" on page 434

ColumnMappings property

Returns a collection of "ULBulkCopyColumnMapping class" on page 445 items. Column mappings define the relationships between columns in the data source and columns in the destination.

Prototypes

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 434
- "ULBulkCopy members" on page 434

DestinationTableName property

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

Prototypes

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 434
- "ULBulkCopy members" on page 434

NotifyAfter property

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

Prototypes

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 434
- "ULBulkCopy members" on page 434
- "BulkCopyTimeout property" on page 438

Close method

Closes the ULBulkCopy instance.

Prototypes

Visual Basic
Public Sub **Close()**

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

See also

- "ULBulkCopy class" on page 434
- "ULBulkCopy members" on page 434

Dispose method

Disposes of the ULBulkCopy instance.

Prototypes

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

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

See also

- "ULBulkCopy class" on page 434
- "ULBulkCopy members" on page 434

WriteToServer methods

Copies all rows in the supplied array of DataRow objects to a destination table specified by the "DestinationTableName property" on page 440 of the ULBulkCopy object.

WriteToServer(DataRow[]) method

Copies all rows in the supplied array of DataRow objects to a destination table specified by the "DestinationTableName property" on page 440 of the ULBulkCopy object.

Prototypes

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

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

Parameters

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

See also

- "ULBulkCopy class" on page 434
- "ULBulkCopy members" on page 434
- "WriteToServer methods" on page 441

WriteToServer(DataTable) method

Copies all rows in the supplied DataTable to a destination table specified by the "DestinationTableName property" on page 440 of the ULBulkCopy object.

Prototypes

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

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

Parameters

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

See also

- "ULBulkCopy class" on page 434
- "ULBulkCopy members" on page 434
- "WriteToServer methods" on page 441

WriteToServer(IDataReader) method

Copies all rows in the supplied IDataReader to a destination table specified by the "DestinationTableName property" on page 440 of the ULBulkCopy object.

Prototypes

Visual Basic

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

C#

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

Parameters

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

See also

- "ULBulkCopy class" on page 434
- "ULBulkCopy members" on page 434
- "WriteToServer methods" on page 441

WriteToServer(DataTable, DataRowState) method

Copies all rows in the supplied DataTable with the specified row state to a destination table specified by the "DestinationTableName property" on page 440 of the ULBulkCopy object.

Prototypes

Visual Basic

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

C#

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

Parameters

- **table** A DataTable whose rows will be copied to the destination table.
- **rowState** A value from the 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 434
- "ULBulkCopy members" on page 434

- "WriteToServer methods" on page 441

ULRowsCopied event

This event occurs every time the number of rows specified by the "NotifyAfter property" on page 440 have been processed.

Prototypes

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 or committed. You cannot call the Close method from this event.

See also

- "ULBulkCopy class" on page 434
- "ULBulkCopy members" on page 434
- "NotifyAfter property" on page 440

ULBulkCopyColumnMapping class

Defines the mapping between a column in a "ULBulkCopy class" on page 434 instance's data source and a column in the instance's destination table. This class cannot be inherited.

Prototypes

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 445

ULBulkCopyColumnMapping members

Public constructors

Member name	Description
ULBulkCopyColumnMapping constructors	Initializes a new instance of the "ULBulkCopyColumnMapping class" on page 445.

Public properties

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

See also

- "ULBulkCopyColumnMapping class" on page 445

ULBulkCopyColumnMapping constructors

Initializes a new instance of the "ULBulkCopyColumnMapping class" on page 445.

ULBulkCopyColumnMapping() constructor

Creates a new column mapping.

Prototypes

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 445
- "ULBulkCopyColumnMapping members" on page 445
- "ULBulkCopyColumnMapping constructors" on page 446

ULBulkCopyColumnMapping(Int32, Int32) constructor

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

Prototypes

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 445
- "ULBulkCopyColumnMapping members" on page 445
- "ULBulkCopyColumnMapping constructors" on page 446

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.

Prototypes

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 445
- "ULBulkCopyColumnMapping members" on page 445
- "ULBulkCopyColumnMapping constructors" on page 446

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.

Prototypes

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 445
- "ULBulkCopyColumnMapping members" on page 445
- "ULBulkCopyColumnMapping constructors" on page 446

ULBulkCopyColumnMapping(String, String) constructor

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

Prototypes

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 445
- "ULBulkCopyColumnMapping members" on page 445
- "ULBulkCopyColumnMapping constructors" on page 446

DestinationColumn property

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

Prototypes

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 property" on page 449 has priority.

Remarks

The DestinationColumn property and "DestinationOrdinal property" on page 449 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 445
- "ULBulkCopyColumnMapping members" on page 445
- "DestinationOrdinal property" on page 449

DestinationOrdinal property

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

Prototypes

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 property" on page 449 and DestinationOrdinal property 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 445
- "ULBulkCopyColumnMapping members" on page 445
- "DestinationColumn property" on page 449

SourceColumn property

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

Prototypes

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 property" on page 451 has priority.

Remarks

The SourceColumn property and "SourceOrdinal property" on page 451 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 445
- "ULBulkCopyColumnMapping members" on page 445

- "SourceOrdinal property" on page 451

SourceOrdinal property

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

Prototypes

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 property" on page 450 and SourceOrdinal property 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 445
- "ULBulkCopyColumnMapping members" on page 445
- "SourceColumn property" on page 450

ULBulkCopyColumnMappingCollection class

A collection of "ULBulkCopyColumnMapping class" on page 445 objects that inherits from System.Collections.CollectionBase. This class cannot be inherited.

Prototypes

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 452

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.
Item property	Gets the "ULBulkCopyColumnMapping class" on page 445 object at the specified index.

Public methods

Member name	Description
Add methods	Adds the specified "ULBulkCopyColumnMapping class" on page 445 to the collection.
Clear (inherited from CollectionBase)	Removes all objects from the CollectionBase instance.
Contains method	Returns whether the specified "ULBulkCopyColumnMapping class" on page 445 object exists in the collection.

Member name	Description
CopyTo method	Copies the elements of the ULBulkCopyColumnMappingCollection to an array of "ULBulkCopyColumnMapping class" on page 445 items, starting at a particular index.
GetEnumerator (inherited from CollectionBase)	Returns an enumerator that iterates through the CollectionBase instance.
IndexOf method	Returns the index of the specified "ULBulkCopyColumnMapping class" on page 445 within the collection.
Remove method	Removes the specified "ULBulkCopyColumnMapping class" on page 445 element from the ULBulkCopyColumnMappingCollection.
RemoveAt method	Removes the mapping at the specified index from the collection.

See also

- "ULBulkCopyColumnMappingCollection class" on page 452

Item property

Gets the "ULBulkCopyColumnMapping class" on page 445 object at the specified index.

Prototypes

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 class" on page 445 object to find.

Property value

An "ULBulkCopyColumnMapping class" on page 445 object is returned.

See also

- "ULBulkCopyColumnMappingCollection class" on page 452
- "ULBulkCopyColumnMappingCollection members" on page 452

Add methods

Adds the specified "ULBulkCopyColumnMapping class" on page 445 to the collection.

Add(ULBulkCopyColumnMapping) method

Adds the specified "ULBulkCopyColumnMapping class" on page 445 to the collection.

Prototypes

Visual Basic

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

C#

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

Parameters

- **bulkCopyColumnMapping** The "ULBulkCopyColumnMapping class" on page 445 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 452
- "ULBulkCopyColumnMappingCollection members" on page 452
- "Add methods" on page 454

Add(Int32, Int32) method

Creates a new "ULBulkCopyColumnMapping class" on page 445 using ordinals to specify both source and destination columns, and adds the mapping to the collection.

Prototypes

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 452
- "ULBulkCopyColumnMappingCollection members" on page 452
- "Add methods" on page 454

Add(Int32, String) method

Creates a new "ULBulkCopyColumnMapping class" on page 445 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.

Prototypes

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 452

- "ULBulkCopyColumnMappingCollection members" on page 452
- "Add methods" on page 454

Add(String, Int32) method

Creates a new "ULBulkCopyColumnMapping class" on page 445 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.

Prototypes

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 452
- "ULBulkCopyColumnMappingCollection members" on page 452
- "Add methods" on page 454

Add(String, String) method

Creates a new "ULBulkCopyColumnMapping class" on page 445 using column names to specify both source and destination columns, and adds the mapping to the collection.

Prototypes

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 452
- "ULBulkCopyColumnMappingCollection members" on page 452
- "Add methods" on page 454

Contains method

Returns whether the specified "ULBulkCopyColumnMapping class" on page 445 object exists in the collection.

Prototypes

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

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

Parameters

- **value** A valid "ULBulkCopyColumnMapping class" on page 445 object.

Return value

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

See also

- "ULBulkCopyColumnMappingCollection class" on page 452
- "ULBulkCopyColumnMappingCollection members" on page 452

CopyTo method

Copies the elements of the ULBulkCopyColumnMappingCollection to an array of "ULBulkCopyColumnMapping class" on page 445 items, starting at a particular index.

Prototypes

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 class" on page 445 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 452
- "ULBulkCopyColumnMappingCollection members" on page 452

IndexOf method

Returns the index of the specified "ULBulkCopyColumnMapping class" on page 445 within the collection.

Prototypes

Visual Basic

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

C#

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

Parameters

- **value** The "ULBulkCopyColumnMapping class" on page 445 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 452
- "ULBulkCopyColumnMappingCollection members" on page 452

Remove method

Removes the specified "ULBulkCopyColumnMapping class" on page 445 element from the ULBulkCopyColumnMappingCollection.

Prototypes

Visual Basic

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

C#

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

Parameters

- **value** The "ULBulkCopyColumnMapping class" on page 445 object to be removed from the collection.

See also

- "ULBulkCopyColumnMappingCollection class" on page 452
- "ULBulkCopyColumnMappingCollection members" on page 452

RemoveAt method

Removes the mapping at the specified index from the collection.

Prototypes

Visual Basic

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

C#

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


Parameters

- **index** The zero-based index of the "ULBulkCopyColumnMapping class" on page 445 object to be removed from the collection.

See also

- "ULBulkCopyColumnMappingCollection class" on page 452
- "ULBulkCopyColumnMappingCollection members" on page 452

ULBulkCopyOptions enumeration

A bitwise flag that specifies one or more options to use with an instance of the "ULBulkCopy class" on page 434.

Prototypes

Visual Basic
Public Enum **ULBulkCopyOptions**

C#
public enum **ULBulkCopyOptions**

Remarks

The ULBulkCopyOptions enumeration is used when you construct a "ULBulkCopy class" on page 434 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

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.

Prototypes

Visual Basic
Public NotInheritable Class **ULCommand**
  Inherits DbCommand

C#
public sealed class **ULCommand** : DbCommand

Remarks

ULCommand objects can be created directly, or with the "CreateCommand method" on page 515. This method ensures that the command has the correct transaction for executing statements on the given connection.

The "Transaction property" on page 472 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
"ExecuteNonQuery method" on page 484	Executes a statement that does not return a result set, such as a SQL INSERT, DELETE, or UPDATE statement.
"ExecuteReader() method" on page 484	Executes a SQL SELECT statement and returns the result set in a "ULDataReader class" on page 602. Use this method for creating read-only result sets.
"ExecuteResultSet() method" on page 486	**UL Ext.:** Executes a SQL SELECT statement and returns the result set in a "ULResultSet class" on page 724. Use this method for creating mutable result sets.
"ExecuteScalar method" on page 488	Executes a SQL SELECT statement and returns a single value.
"ExecuteTable() method" on page 489	**UL Ext.:** Retrieves a database table in a "ULTable class" on page 830 for direct manipulation. The "CommandText property" on page 467 is interpreted as the name of the table and the "IndexName property" on page 470 can be used to specify a table sorting order. The "CommandType property" on page 468 must be CommandType.TableDirect.

You can reset most properties, including the "CommandText property" on page 467, 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 Component.Dispose or

explicitly call the Component.Dispose. In Visual Basic, you always explicitly call the Component.Dispose.

Inherits: DbCommand

Implements: IDbCommand, IDisposable

See also

- "ULCommand members" on page 463

ULCommand members

Public constructors

Member name	Description
ULCommand constructors	Initializes a new instance of the "ULCommand class" on page 462.

Public properties

Member name	Description
CommandText property	Specifies the text of the SQL statement or the name of the table when the "CommandType property" on page 468 is CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.
CommandTimeout property	This feature is not supported by UltraLite.NET.
CommandType property	Specifies the type of command to be executed.
Connection property	The connection object on which to execute the ULCommand object.
DesignTimeVisible property	Indicates if the ULCommand should be visible in a customized Windows Form Designer control.
IndexName property	**UL Ext.:** Specifies the name of the index to open (sort) the table with when the "CommandType property" on page 468 is CommandType.TableDirect.
Parameters property	Specifies the parameters for the current statement.
Plan property	**UL Ext.:** Returns the access plan UltraLite.NET uses to execute a query. This property is intended primarily for use during development.
Transaction property	Specifies the "ULTransaction class" on page 862 in which the ULCommand executes.
UpdatedRowSource property	Specifies how command results are applied to the DataRow when used by the Update method of the ULDataAdapter.

Public methods

Member name	Description
BeginExecuteNonQuery methods	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, given a callback procedure and state information.
BeginExecuteReader methods	Initiates the asynchronous execution of a SQL statement that is described by this ULCommand, and retrieves the result set.
Cancel method	This method is not supported in UltraLite.NET.
CreateParameter method	Provides a "ULParameter class" on page 687 object for supplying parameters to ULCommand objects.
EndExecuteNonQuery method	Finishes asynchronous execution of a SQL statement.
EndExecuteReader method	Finishes asynchronous execution of a SQL statement, returning the requested "ULDataReader class" on page 602.
ExecuteNonQuery method	Executes a statement that does not return a result set, such as a SQL INSERT, DELETE, or UPDATE statement.
ExecuteReader methods	Executes a SQL SELECT statement and returns the result set.
ExecuteResultSet methods	**UL Ext.:** Executes a SQL SELECT statement and returns the result set as a "ULResultSet class" on page 724.
ExecuteScalar method	Executes a SQL SELECT statement and returns a single value.
ExecuteTable methods	**UL Ext.:** Retrieves in a "ULTable class" on page 830 a database table for direct manipulation. The "CommandText property" on page 467 is interpreted as the name of the table and "IndexName property" on page 470 can be used to specify a table sorting order.
Prepare method	Pre-compiles and stores the SQL statement of this command.

See also

- "ULCommand class" on page 462

ULCommand constructors

Initializes a new instance of the "ULCommand class" on page 462.

ULCommand() constructor

Initializes a ULCommand object.

Prototypes

Visual Basic
Public Sub **New()**

C#
public **ULCommand();**

Remarks

The ULCommand object needs to have the "CommandText property" on page 467, "Connection property" on page 469, and "Transaction property" on page 472 set before a statement can be executed.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ULCommand constructors" on page 464
- "CreateCommand method" on page 515
- "ULCommand(String) constructor" on page 465
- "ULCommand(String, ULConnection) constructor" on page 466
- "ULCommand(String, ULConnection, ULTransaction) constructor" on page 466

ULCommand(String) constructor

Initializes a ULCommand object with the specified command text.

Prototypes

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 "CommandType property" on page 468 is CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.

Remarks

The ULCommand object needs to have the "Connection property" on page 469 and "Transaction property" on page 472 set before a statement can be executed.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ULCommand constructors" on page 464

- "CreateCommand method" on page 515
- "ULCommand() constructor" on page 464
- "ULCommand(String, ULConnection) constructor" on page 466
- "ULCommand(String, ULConnection, ULTransaction) constructor" on page 466

ULCommand(String, ULConnection) constructor

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

Prototypes

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 "CommandType property" on page 468 is CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.
- **connection** The "ULConnection class" on page 498 object representing the current connection.

Remarks

The ULCommand object may need to have the "Transaction property" on page 472 set before a statement can be executed.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ULCommand constructors" on page 464
- "CreateCommand method" on page 515
- "ULCommand() constructor" on page 464
- "ULCommand(String) constructor" on page 465
- "ULCommand(String, ULConnection, ULTransaction) constructor" on page 466

ULCommand(String, ULConnection, ULTransaction) constructor

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

Prototypes

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 "CommandType property" on page 468 is CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.
- **connection** The "ULConnection class" on page 498 object representing the current connection.
- **transaction** The "ULTransaction class" on page 862 in which the ULCommand executes.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ULCommand constructors" on page 464
- "CreateCommand method" on page 515
- "ULCommand() constructor" on page 464
- "ULCommand(String) constructor" on page 465
- "ULCommand(String, ULConnection) constructor" on page 466

CommandText property

Specifies the text of the SQL statement or the name of the table when the "CommandType property" on page 468 is CommandType.TableDirect. For parameterized statements, use a question mark (?) placeholder to pass parameters.

Prototypes

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

It is recommended that SELECT statements used to create read-only result sets ("ExecuteReader() method" on page 484 or "ExecuteScalar method" on page 488) should end with " FOR READ ONLY". For some statements that use temporary tables, there may be a significant performance improvement.

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 462
- "ULCommand members" on page 463
- "ExecuteNonQuery method" on page 484
- "ExecuteReader() method" on page 484
- "ExecuteResultSet() method" on page 486
- "ExecuteScalar method" on page 488
- "ExecuteTable() method" on page 489

CommandTimeout property

This feature is not supported by UltraLite.NET.

Prototypes

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 462
- "ULCommand members" on page 463

CommandType property

Specifies the type of command to be executed.

Prototypes

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

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

Property value

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

Remarks

Supported command types are as follows:

- CommandType.TableDirect - **UL Ext.:** When you specify this CommandType, the "CommandText property" on page 467 must be the name of a database table. You can also specify the index used to open (sort) the table with "IndexName property" on page 470. Use "ExecuteTable() method" on page 489 or "ExecuteReader() method" on page 484 to access the table.
- CommandType.Text - When you specify this CommandType, the "CommandText property" on page 467 must be a SQL statement or query. Use "ExecuteNonQuery method" on page 484 to execute a non-query SQL statement and use either "ExecuteReader() method" on page 484 or "ExecuteScalar method" on page 488 to execute a query.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463

Connection property

The connection object on which to execute the ULCommand object.

Prototypes

Visual Basic
Public Property **Connection** As ULConnection

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

Property value

The "ULConnection class" on page 498 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 IDbCommand.Connection and DbCommand.Connection.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463

DesignTimeVisible property

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

Prototypes

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 462
- "ULCommand members" on page 463

IndexName property

UL Ext.: Specifies the name of the index to open (sort) the table with when the "CommandType property" on page 468 is CommandType.TableDirect.

Prototypes

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 462
- "ULCommand members" on page 463
- "ExecuteTable() method" on page 489
- "ExecuteReader() method" on page 484

Parameters property

Specifies the parameters for the current statement.

Prototypes

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

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

Property value

A "ULParameterCollection class" on page 702 holding the parameters of the SQL statement. The default value is the empty collection.

Remarks

Use question marks in the "CommandText property" on page 467 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 "CommandText property" on page 467 as there are parameters in this collection.

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

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ULParameter class" on page 687

Plan property

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

Prototypes

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 462
- "ULCommand members" on page 463

Transaction property

Specifies the "ULTransaction class" on page 862 in which the ULCommand executes.

Prototypes

Visual Basic
Public Property **Transaction** As ULTransaction

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

Property value

The "ULTransaction class" on page 862 in which the ULCommand executes. This should be the current transaction of the connection specified by the "Connection property" on page 469. 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 IDbCommand.Transaction and DbCommand.Transaction.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "BeginTransaction() method" on page 510

UpdatedRowSource property

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

Prototypes

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

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

Property value

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

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463

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

Prototypes

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

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

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "BeginExecuteNonQuery methods" on page 473

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.

Prototypes

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 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 IAsyncResult.AsyncState.

Return value

An IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking "EndExecuteNonQuery method" on page 478, which returns the number of affected rows.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "BeginExecuteNonQuery methods" on page 473
- "EndExecuteNonQuery method" on page 478

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.

Prototypes

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

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

Return value

An IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking "EndExecuteReader method" on page 481, which returns an "ULDataReader class" on page 602 instance that can be used to retrieve the returned rows.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "BeginExecuteReader methods" on page 474
- "EndExecuteReader method" on page 481

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.

Prototypes

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

C#
```
public IAsyncResult BeginExecuteReader(
  CommandBehavior cmdBehavior
);
```

Parameters

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

Return value

An IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking "EndExecuteReader method" on page 481, which returns an "ULDataReader class" on page 602 instance that can be used to retrieve the returned rows.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "BeginExecuteReader methods" on page 474
- "EndExecuteReader method" on page 481

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.

Prototypes

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 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 IAsyncResult.AsyncState.

Return value

An IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking "EndExecuteReader method" on page 481, which returns an "ULDataReader class" on page 602 instance that can be used to retrieve the returned rows.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "BeginExecuteReader methods" on page 474
- "EndExecuteReader method" on page 481

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.

Prototypes

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

Return value

An IAsyncResult that can be used to poll, wait for results, or both is returned; this value is also needed when invoking "EndExecuteReader method" on page 481, which returns an "ULDataReader class" on page 602 instance that can be used to retrieve the returned rows.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "BeginExecuteReader methods" on page 474
- "EndExecuteReader method" on page 481

Cancel method

This method is not supported in UltraLite.NET.

Prototypes

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 462
- "ULCommand members" on page 463

CreateParameter method

Provides a "ULParameter class" on page 687 object for supplying parameters to ULCommand objects.

Prototypes

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

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

Return value

A new parameter, as a "ULParameter class" on page 687 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 class" on page 687 object. You can set properties on the ULParameter to specify the value for the parameter.

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

See also

- "ULCommand class" on page 462

- "ULCommand members" on page 463

EndExecuteNonQuery method

Finishes asynchronous execution of a SQL statement.

Prototypes

Visual Basic

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

C#

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

Parameters

- **asyncResult** The 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 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 462
- "ULCommand members" on page 463
- "BeginExecuteNonQuery() method" on page 473

EndExecuteReader method

Finishes asynchronous execution of a SQL statement, returning the requested "ULDataReader class" on page 602.

Prototypes

Visual Basic

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

C#

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

Parameters

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

Return value

An "ULDataReader class" on page 602 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 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 462
- "ULCommand members" on page 463
- "BeginExecuteReader() method" on page 474

ExecuteNonQuery method

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

Prototypes

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 "CommandText property" on page 467 and "Parameters property" on page 471 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 "CommandType property" on page 468 cannot be CommandType.TableDirect.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463

ExecuteReader methods

Executes a SQL SELECT statement and returns the result set.

ExecuteReader() method

Executes a SQL SELECT statement and returns the result set.

Prototypes

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

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

Return value

The result set as a "ULDataReader class" on page 602 object.

Remarks

The statement is the current ULCommand object, with the "CommandText property" on page 467 and any "Parameters property" on page 471 as required. The "ULDataReader class" on page 602 object is a read-only result set. For editable result sets, use "ExecuteResultSet() method" on page 486, "ExecuteTable() method" on page 489, or a "ULDataAdapter class" on page 574.

If the "CommandType property" on page 468 is CommandType.TableDirect, ExecuteReader performs an "ExecuteTable() method" on page 489 and returns a "ULTable class" on page 830 downcast as a "ULDataReader class" on page 602.

It is recommended that SELECT statements used with this method to create read-only result sets end with FOR READ ONLY. For some statements that use temporary tables, there may be a significant performance improvement.

This is the strongly-typed version of IDbCommand.ExecuteReader and DbCommand.ExecuteReader.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ExecuteReader methods" on page 484
- "ExecuteReader(CommandBehavior) method" on page 485

ExecuteReader(CommandBehavior) method

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

Prototypes

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

C#
```
public ULDataReader ExecuteReader(
  CommandBehavior cmdBehavior
);
```


Parameters

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

Return value

The result set as a "ULDataReader class" on page 602 object.

Remarks

The statement is the current ULCommand object, with the "CommandText property" on page 467 and any "Parameters property" on page 471 as required. The "ULDataReader class" on page 602 object is a read-only result set. For editable result sets, use "ExecuteResultSet(CommandBehavior) method" on page 487, "ExecuteTable(CommandBehavior) method" on page 489, or a "ULDataAdapter class" on page 574.

If the "CommandType property" on page 468 is CommandType.TableDirect, ExecuteReader performs an "ExecuteTable(CommandBehavior) method" on page 489 and returns a "ULTable class" on page 830 downcast as a "ULDataReader class" on page 602.

It is recommended that SELECT statements used with this method to create read-only result sets end with FOR READ ONLY. For some statements that use temporary tables, there may be a significant performance improvement.

This is the strongly-typed version of IDbCommand.ExecuteReader and DbCommand.ExecuteReader.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ExecuteReader methods" on page 484
- "ExecuteReader() method" on page 484

ExecuteResultSet methods

UL Ext.: Executes a SQL SELECT statement and returns the result set as a "ULResultSet class" on page 724.

ExecuteResultSet() method

UL Ext.: Executes a SQL SELECT statement and returns the result set as a "ULResultSet class" on page 724.

Prototypes

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

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

Return value

The result set as a "ULResultSet class" on page 724 object.

Remarks

The statement is the current ULCommand object, with the "CommandText property" on page 467 and any "Parameters property" on page 471 as required. The "ULResultSet class" on page 724 object is an editable result set on which you can perform positioned updates and deletes. For fully editable result sets, use "ExecuteTable() method" on page 489 or a "ULDataAdapter class" on page 574.

If the "CommandType property" on page 468 is CommandType.TableDirect, ExecuteReader performs an "ExecuteTable() method" on page 489 and returns a "ULTable class" on page 830 downcast as a "ULResultSet class" on page 724.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ExecuteResultSet methods" on page 486
- "ExecuteResultSet(CommandBehavior) method" on page 487

ExecuteResultSet(CommandBehavior) method

UL Ext.: Executes a SQL SELECT statement with the specified command behavior and returns the result set as a "ULResultSet class" on page 724.

Prototypes

Visual Basic

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

C#

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

Parameters

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

Return value

The result set as a "ULResultSet class" on page 724 object.

Remarks

The statement is the current ULCommand object, with the "CommandText property" on page 467 and any "Parameters property" on page 471 as required. The "ULResultSet class" on page 724 object is an editable result set on which you can perform positioned updates and deletes. For fully editable result sets, use "ExecuteTable(CommandBehavior) method" on page 489 or a "ULDataAdapter class" on page 574.

If the "CommandType property" on page 468 is CommandType.TableDirect, ExecuteReader performs an "ExecuteTable(CommandBehavior) method" on page 489 and returns a "ULTable class" on page 830 downcast as a "ULResultSet class" on page 724.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ExecuteResultSet methods" on page 486
- "ExecuteReader() method" on page 484

ExecuteScalar method

Executes a SQL SELECT statement and returns a single value.

Prototypes

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 "CommandText property" on page 467 and any "Parameters property" on page 471 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 "CommandType property" on page 468 is CommandType.TableDirect, ExecuteScalar performs an "ExecuteTable() method" on page 489 and returns the first column of the first row.

It is recommended that SELECT statements used with this method to create read-only result sets end with FOR READ ONLY. For some statements that use temporary tables, there may be a significant performance improvement.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463

ExecuteTable methods

UL Ext.: Retrieves in a "ULTable class" on page 830 a database table for direct manipulation. The "CommandText property" on page 467 is interpreted as the name of the table and "IndexName property" on page 470 can be used to specify a table sorting order.

ExecuteTable() method

UL Ext.: Retrieves in a "ULTable class" on page 830 a database table for direct manipulation. The "CommandText property" on page 467 is interpreted as the name of the table and "IndexName property" on page 470 can be used to specify a table sorting order.

Prototypes

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

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

Return value

The table as a "ULTable class" on page 830 object.

Remarks

The "CommandType property" on page 468 must be set to CommandType.TableDirect.

If the "IndexName property" on page 470 is a null reference (Nothing in Visual Basic), the primary key is used to open the table. Otherwise, the table is opened using the "IndexName property" on page 470 value as the name of the index by which to sort.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ExecuteTable methods" on page 489
- "ExecuteTable(CommandBehavior) method" on page 489

ExecuteTable(CommandBehavior) method

UL Ext.: Retrieves, with the specified command behavior, a database table for direct manipulation. The "CommandText property" on page 467 is interpreted as the name of the table and "IndexName property" on page 470 can be used to specify a table sorting order.

Prototypes

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


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

Parameters

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

Return value

The table as a "ULTable class" on page 830 object.

Remarks

The "CommandType property" on page 468 must be set to CommandType.TableDirect.

If the "IndexName property" on page 470 is a null reference (Nothing in Visual Basic), the primary key is used to open the table. Otherwise, the table is opened using the "IndexName property" on page 470 value as the name of the index by which to sort.

See also

- "ULCommand class" on page 462
- "ULCommand members" on page 463
- "ExecuteTable methods" on page 489
- "ExecuteTable() method" on page 489

Prepare method

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

Prototypes

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 462
- "ULCommand members" on page 463

ULCommandBuilder class

Automatically generates single-table commands used to reconcile changes made to aDataSet with the associated database. This class cannot be inherited.

Prototypes

Visual Basic
Public NotInheritable Class **ULCommandBuilder**
  Inherits Component

C#
public sealed class **ULCommandBuilder** : Component

Remarks

The "ULDataAdapter class" on page 574 does not automatically generate the SQL statements required to reconcile changes made to a 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 class" on page 574. Then, any additional SQL statements that you do not set are generated by the ULCommandBuilder.

Inherits: Component

Implements: IDisposable

Example

The following example uses the "ULCommand class" on page 462, along with "ULDataAdapter class" on page 574 and "ULConnection class" on page 498, 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 492

ULCommandBuilder members

Public constructors

Member name	Description
ULCommandBuilder constructors	Initializes a new instance of the "ULCommandBuilder class" on page 491.

Public properties

Member name	Description
DataAdapter property	Gets or sets a "ULDataAdapter class" on page 574 object for which SQL statements are automatically generated.
QuotePrefix property	Gets or sets the starting character or characters to use when specifying UltraLite database objects, such as tables or columns, whose names contain characters such as spaces or reserved tokens.
QuoteSuffix property	Gets or sets the ending character or characters to use when specifying UltraLite database objects, such as tables or columns, whose names contain characters such as spaces or reserved tokens.

Public methods

Member name	Description
GetDeleteCommand method	Gets the automatically generated "ULCommand class" on page 462 object required to perform deletions on the database.
GetInsertCommand method	Gets the automatically generated "ULCommand class" on page 462 object required to perform insertions on the database.
GetUpdateCommand method	Gets the automatically generated "ULCommand class" on page 462 object required to perform updates on the database.
RefreshSchema method	Clears the commands associated with this ULCommandBuilder.

See also

- "ULCommandBuilder class" on page 491

ULCommandBuilder constructors

Initializes a new instance of the "ULCommandBuilder class" on page 491.

ULCommandBuilder() constructor

Initializes a ULCommandBuilder object.

Prototypes

Visual Basic
Public Sub **New()**

C#
public **ULCommandBuilder();**

See also

- "ULCommandBuilder class" on page 491
- "ULCommandBuilder members" on page 492
- "ULCommandBuilder constructors" on page 493
- "ULCommandBuilder(ULDataAdapter) constructor" on page 493

ULCommandBuilder(ULDataAdapter) constructor

Initializes a ULCommandBuilder object with the specified "ULDataAdapter class" on page 574 object.

Prototypes

Visual Basic
Public Sub **New(_**


```
  ByVal adapter As ULDataAdapter _
)

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

Parameters

- **adapter** A "ULDataAdapter class" on page 574 object.

See also

- "ULCommandBuilder class" on page 491
- "ULCommandBuilder members" on page 492
- "ULCommandBuilder constructors" on page 493

DataAdapter property

Gets or sets a "ULDataAdapter class" on page 574 object for which SQL statements are automatically generated.

Prototypes

```
Visual Basic
Public Property DataAdapter As ULDataAdapter

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

Property value

A "ULDataAdapter class" on page 574 object.

See also

- "ULCommandBuilder class" on page 491
- "ULCommandBuilder members" on page 492

QuotePrefix property

Gets or sets the starting character or characters to use when specifying UltraLite database objects, such as tables or columns, whose names contain characters such as spaces or reserved tokens.

Prototypes

```
Visual Basic
Public Property QuotePrefix As String

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


Property value

The starting character or characters to use. The default is an empty string.

See also

- "ULCommandBuilder class" on page 491
- "ULCommandBuilder members" on page 492
- "QuoteSuffix property" on page 495

QuoteSuffix property

Gets or sets the ending character or characters to use when specifying UltraLite database objects, such as tables or columns, whose names contain characters such as spaces or reserved tokens.

Prototypes

Visual Basic
Public Property **QuoteSuffix** As String

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

Property value

The ending character or characters to use. The default is an empty string.

See also

- "ULCommandBuilder class" on page 491
- "ULCommandBuilder members" on page 492
- "QuotePrefix property" on page 494

GetDeleteCommand method

Gets the automatically generated "ULCommand class" on page 462 object required to perform deletions on the database.

Prototypes

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

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

Return value

The automatically generated "ULCommand class" on page 462 object required to perform deletions.

Remarks

After the SQL statement is first generated, the application must explicitly call the "RefreshSchema method" on page 497 if it changes the "SelectCommand property" on page 580 in any way. Otherwise, the

GetDeleteCommand 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 or the GetDeleteCommand method.

See also

- "ULCommandBuilder class" on page 491
- "ULCommandBuilder members" on page 492

GetInsertCommand method

Gets the automatically generated "ULCommand class" on page 462 object required to perform insertions on the database.

Prototypes

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

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

Return value

The automatically generated "ULCommand class" on page 462 object required to perform insertions.

Remarks

After the SQL statement is first generated, the application must explicitly call the "RefreshSchema method" on page 497 if it changes the "SelectCommand property" on page 580 in any way. Otherwise, the GetInsertCommand 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 or the GetInsertCommand method.

See also

- "ULCommandBuilder class" on page 491
- "ULCommandBuilder members" on page 492

GetUpdateCommand method

Gets the automatically generated "ULCommand class" on page 462 object required to perform updates on the database.

Prototypes

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

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

Return value

The automatically generated "ULCommand class" on page 462 object required to perform updates.

Remarks

After the SQL statement is first generated, the application must explicitly call the "RefreshSchema method" on page 497 if it changes the "SelectCommand property" on page 580 in any way. Otherwise, the GetUpdateCommand 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 or the GetUpdateCommand method.

See also

- "ULCommandBuilder class" on page 491
- "ULCommandBuilder members" on page 492

RefreshSchema method

Clears the commands associated with this ULCommandBuilder.

Prototypes

Visual Basic
Public Sub **RefreshSchema()**

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

Remarks

After the SQL statement is first generated, the application must explicitly call "RefreshSchema method" on page 497 if it changes the "SelectCommand property" on page 580statement in any way. Otherwise, "GetInsertCommand method" on page 496, "GetDeleteCommand method" on page 495, and "GetUpdateCommand method" on page 496 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 DbDataAdapter.Update , "GetInsertCommand method" on page 496, "GetDeleteCommand method" on page 495, or "GetUpdateCommand method" on page 496.

See also

- "ULCommandBuilder class" on page 491
- "ULCommandBuilder members" on page 492

ULConnection class

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

Prototypes

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 "RuntimeType property" on page 586 to the appropriate value before using any other UltraLite.NET API.

A connection to an existing database is opened using the "Open method" on page 524.

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 "Schema property" on page 507.

Implements: IDbConnection, IDisposable

See also

- "ULConnection members" on page 498

ULConnection members

Public constructors

Member name	Description
ULConnection constructors	Initializes a new instance of the "ULConnection class" on page 498.

Public fields

Member name	Description
INVALID_DATABASE_ID field	**UL Ext.:** A database ID constant indicating that the "DatabaseID property" on page 505 has not been set. This field is constant and read-only.

Public properties

Member name	Description
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 class" on page 531 object.
ConnectionTimeout property	This feature is not supported by UltraLite.NET.
DataSource property	This feature is not supported by UltraLite.NET.
Database property	Returns the name of the database to which the connection opens.
DatabaseID property	**UL Ext.:** Specifies the Database ID value to be used for global autoincrement columns.
DatabaseManager property	**UL Ext.:** Provides access to the singleton "ULDatabaseManager class" on page 585 object.
GlobalAutoIncrementUsage property	**UL Ext.:** Returns the percentage of available global autoincrement values that have been used.
LastIdentity property	**UL Ext.:** Returns the most recent identity value used.
Schema property	**UL Ext.:** Provides access to the schema of the current database associated with this connection.
ServerVersion property	This feature is not supported by UltraLite.NET.
State property	Returns the current state of the connection.
SyncParms property	**UL Ext.:** Specifies the synchronization settings for this connection.
SyncResult property	**UL Ext.:** Returns the results of the last synchronization for this connection.

Public methods

Member name	Description
BeginTransaction methods	Returns a transaction object. Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a "Commit method" on page 864 or "Rollback method" on page 864.
ChangeDatabase method	Changes the current database for an open ULConnection.
ChangeEncryptionKey method	**UL Ext.:** Changes the database's encryption key to the specified new key.
ChangePassword method	Changes the password for the user indicated in the connection string to the supplied new password.
Close method	Closes the database connection.

Member name	Description
CountUploadRows methods	**UL Ext.:** Returns the number of rows that need to be uploaded when the next synchronization takes place.
CreateCommand method	Creates and initializes a "ULCommand class" on page 462 object associated with this connection and its current transaction. You can use the properties of the ULCommand to control its behavior.
EnlistTransaction (inherited from DbConnection)	Enlists in the specified transaction.
ExecuteTable methods	**UL Ext.:** Retrieves in a "ULTable class" on page 830 a database table for direct manipulation. The table is opened (sorted) using the table's primary key.
GetLastDownloadTime method	**UL Ext.:** Returns the time of the most recent download of the specified publication.
GetNewUUID method	**UL Ext.:** Generates a new UUID (Guid).
GetSchema methods	Returns schema information for the data source of this DbConnection.
GrantConnectTo method	**UL Ext.:** Grants access to an UltraLite database for a user ID with a specified password.
Open method	Opens a connection to a database using the previously-specified connection string.
ResetLastDownloadTime method	**UL Ext.:** Resets the time of the most recent download.
RevokeConnectFrom method	**UL Ext.:** Revokes access to an UltraLite database from the specified user ID.
RollbackPartialDownload method	**UL Ext.:** Rolls back outstanding changes to the database from a partial download.
StartSynchronizationDelete method	**UL Ext.:** Marks all subsequent deletes made by this connection for synchronization.
StopSynchronizationDelete method	**UL Ext.:** Prevents delete operations from being synchronized.
Synchronize methods	**UL Ext.:** Synchronize the database using the current "SyncParms property" on page 509.

Public events

Member name	Description
InfoMessage event	Occurs when UltraLite.NET sends a warning or an informational message on this connection.

Member name	Description
StateChange event	Occurs when this connection changes state.

See also

- "ULConnection class" on page 498

ULConnection constructors

Initializes a new instance of the "ULConnection class" on page 498.

ULConnection() constructor

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

Prototypes

Visual Basic
Public Sub **New()**

C#
public **ULConnection();**

Remarks

To use the UltraLite Engine runtime of UltraLite.NET, set "RuntimeType property" on page 586 to the appropriate value before using any other UltraLite.NET API.

The ULConnection object needs to have the "ConnectionString property" on page 503 set before it can be opened.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "ULConnection constructors" on page 501
- "Open method" on page 524
- "ULConnection(String) constructor" on page 501

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.

Prototypes

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 the "ConnectionString property" on page 503.

Remarks

To use the UltraLite Engine runtime of UltraLite.NET, set "RuntimeType property" on page 586 to the appropriate value before using any other UltraLite.NET API.

The connection string can be supplied using a "ULConnectionParms class" on page 531 object.

Example

The following code creates and opens a connection to the existing database \UltraLite\MyDatabase.udb on a Windows CE 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 498
- "ULConnection members" on page 498
- "ULConnection constructors" on page 501
- "Open method" on page 524
- "ULConnection() constructor" on page 501

INVALID_DATABASE_ID field

UL Ext.: A database ID constant indicating that the "DatabaseID property" on page 505 has not been set. This field is constant and read-only.

Prototypes

Visual Basic
Public Shared **INVALID_DATABASE_ID** As Long

C#
public const long **INVALID_DATABASE_ID** ;

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498

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 class" on page 531 object.

Prototypes

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 String Parameters Reference" [*UltraLite - Database Management and Reference*].

Parameter values may be enclosed in single (') or double (") quotes. Values that begin with either quote character, values that contain semi-colons (;), and values that need to preserve leading and/or trailing spaces must be quoted. Quoted values may not contain the character used for quoting.

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 method" on page 523) and remove the DBA user (using "RevokeConnectFrom method" on page 525).

Example

The following code creates and opens a connection to the existing database \UltraLite\MyDatabase.udb on a Windows CE 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 498
- "ULConnection members" on page 498
- "Open method" on page 524

ConnectionTimeout property

This feature is not supported by UltraLite.NET.

Prototypes

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 498
- "ULConnection members" on page 498

DataSource property

This feature is not supported by UltraLite.NET.

Prototypes

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 498

- "ULConnection members" on page 498

Database property

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

Prototypes

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 CE 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 498
- "ULConnection members" on page 498

DatabaseID property

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

Prototypes

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,UInt32.MaxValue]. A value of "INVALID_DATABASE_ID field" on page 502 is used to indicate that the database ID has not been set for the current database.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "GetDatabaseProperty method" on page 595

- "SetDatabaseOption method" on page 600

DatabaseManager property

UL Ext.: Provides access to the singleton "ULDatabaseManager class" on page 585 object.

Prototypes

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

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

Property value

A reference to the singleton "ULDatabaseManager class" on page 585 object.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498

GlobalAutoIncrementUsage property

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

Prototypes

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 "DatabaseID property" on page 505.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498

LastIdentity property

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

Prototypes

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 498
- "ULConnection members" on page 498

Schema property

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

Prototypes

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

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

Property value

A reference to the "ULDatabaseSchema class" on page 592 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 498
- "ULConnection members" on page 498

ServerVersion property

This feature is not supported by UltraLite.NET.

Prototypes

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 498
- "ULConnection members" on page 498

State property

Returns the current state of the connection.

Prototypes

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

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

Property value

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

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "StateChange event" on page 529

SyncParms property

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

Prototypes

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

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

Property value

A reference to the "ULSyncParms class" on page 796 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 498
- "ULConnection members" on page 498
- "Synchronize() method" on page 527
- "SyncResult property" on page 509

SyncResult property

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

Prototypes

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

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

Property value

A reference to the "ULSyncResult class" on page 824 object representing the results of the last synchronization for this connection.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "Synchronize() method" on page 527
- "SyncParms property" on page 509

BeginTransaction methods

Returns a transaction object. Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a "Commit method" on page 864 or "Rollback method" on page 864.

BeginTransaction() method

Returns a transaction object. Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a "Commit method" on page 864 or "Rollback method" on page 864.

Prototypes

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

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

Return value

A "ULTransaction class" on page 862 object representing the new transaction.

Remarks

To associate a command with a transaction object, use the "Transaction property" on page 472. The current transaction is automatically associated to commands created by "CreateCommand method" on page 515.

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 until the next call to BeginTransaction.

This is the strongly-typed version of IDbConnection.BeginTransaction and DbConnection.BeginTransaction.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "BeginTransaction methods" on page 510
- "BeginTransaction(IsolationLevel) method" on page 510

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 a "Commit method" on page 864 or "Rollback method" on page 864.

Prototypes

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 IsolationLevel.ReadUncommitted.

Return value

A "ULTransaction class" on page 862 object representing the new transaction.

Remarks

To associate a command with a transaction object, use the "Transaction property" on page 472. The current transaction is automatically associated to commands created by "CreateCommand method" on page 515.

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 until the next call to BeginTransaction.

This is the strongly-typed version of IDbConnection.BeginTransaction and DbConnection.BeginTransaction.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "BeginTransaction methods" on page 510
- "BeginTransaction() method" on page 510

ChangeDatabase method

Changes the current database for an open ULConnection.

Prototypes

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 498
- "ULConnection members" on page 498
- "ConnectionString property" on page 503

ChangeEncryptionKey method

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

Prototypes

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 498
- "ULConnection members" on page 498
- "EncryptionKey property" on page 538

ChangePassword method

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

Prototypes

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 498
- "ULConnection members" on page 498

Close method

Closes the database connection.

Prototypes

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 498
- "ULConnection members" on page 498

CountUploadRows methods

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

CountUploadRows(Int32, UInt32) method

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

Prototypes

Visual Basic

```
Public Function CountUploadRows( _
   ByVal mask As Integer, _
   ByVal threshold As UInt32 _
) As UInt32
```

C#

```
public uint CountUploadRows(
   int mask,
   uint threshold
);
```

Parameters

- **mask** The set of publications to check for rows. For more information, see the "ULPublicationSchema class" on page 720.
- **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 498
- "ULConnection members" on page 498
- "CountUploadRows methods" on page 513
- "CountUploadRows(Int32, Int64) method" on page 514

CountUploadRows(Int32, Int64) method

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

Prototypes

Visual Basic

```
Public Function CountUploadRows( _
   ByVal mask As Integer, _
   ByVal threshold As Long _
) As Long
```

C#

```
public long CountUploadRows(
   int mask,
   long threshold
);
```

Parameters

- **mask** The set of publications to check for rows. For more information, see the "ULPublicationSchema class" on page 720.

- **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 [0,0x0fffffff].

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 498
- "ULConnection members" on page 498
- "CountUploadRows methods" on page 513
- "CountUploadRows(Int32, UInt32) method" on page 513

CreateCommand method

Creates and initializes a "ULCommand class" on page 462 object associated with this connection and its current transaction. You can use the properties of the ULCommand to control its behavior.

Prototypes

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

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

Return value

A new "ULCommand class" on page 462 object.

Remarks

You must set the "CommandText property" on page 467 before the command can be executed.

This is the strongly-typed version of IDbConnection.CreateCommand and DbConnection.CreateCommand.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498

ExecuteTable methods

UL Ext.: Retrieves in a "ULTable class" on page 830 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 class" on page 830 a database table for direct manipulation. The table is opened (sorted) using the table's primary key.

Prototypes

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 class" on page 830 object.

Remarks

This method is a shortcut for the "ExecuteTable() method" on page 489 that does not require a "ULCommand class" on page 462 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 498
- "ULConnection members" on page 498
- "ExecuteTable methods" on page 515
- "ExecuteTable(String, String) method" on page 517
- "ExecuteTable(String, String, CommandBehavior) method" on page 518

ExecuteTable(String, String) method

UL Ext.: Retrieves in a "ULTable class" on page 830 a database table for direct manipulation. The table is opened (sorted) using the specified index.

Prototypes

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 class" on page 830 object.

Remarks

This method is a shortcut for the "ExecuteTable() method" on page 489 that does not require a "ULCommand class" on page 462 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 498
- "ULConnection members" on page 498
- "ExecuteTable methods" on page 515
- "ExecuteTable(String) method" on page 516
- "ExecuteTable(String, String, CommandBehavior) method" on page 518

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.

Prototypes

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 CommandBehavior flags describing the results of the query and its effect on the connection. UltraLite.NET respects only the CommandBehavior.Default, CommandBehavior.CloseConnection, and CommandBehavior.SchemaOnly flags.

Return value

The table as a "ULTable class" on page 830 object.

Remarks

This method is a shortcut for the "ExecuteTable(CommandBehavior) method" on page 489 that does not require a "ULCommand class" on page 462 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 498
- "ULConnection members" on page 498
- "ExecuteTable methods" on page 515
- "ExecuteTable(String) method" on page 516
- "ExecuteTable(String, String) method" on page 517

GetLastDownloadTime method

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

Prototypes

Visual Basic
```
Public Function GetLastDownloadTime( _
  ByVal mask As Integer _
) As Date
```

C#
```
public DateTime GetLastDownloadTime(
  int mask
);
```

Parameters

- **mask** The mask of the publication to check. For more information, see the "ULPublicationSchema class" on page 720.

Return value

The timestamp of the last download.

Remarks

The parameter

mask must reference a single publication or be the special constant "SYNC_ALL_DB field" on page 721 for the time of the last download of the full database.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "ResetLastDownloadTime method" on page 525

GetNewUUID method

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

Prototypes

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

C#
```
public Guid GetNewUUID();
```

Return value

A new UUID as a Guid.

Remarks

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

See also

- "ULConnection class" on page 498

- "ULConnection members" on page 498

GetSchema methods

Returns schema information for the data source of this DbConnection.

GetSchema() method

Returns the list of supported schema collections.

Prototypes

Visual Basic
```
Public Overrides Function GetSchema() As DataTable
```

C#
```
public override DataTable GetSchema();
```

Remarks

See "GetSchema(String, String[]) method" on page 522 for a description of the available metadata.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "GetSchema methods" on page 521

GetSchema(String) method

Returns information for the specified metadata collection for this "ULConnection class" on page 498.

Prototypes

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

See "GetSchema(String, String[]) method" on page 522 for a description of the available metadata.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "GetSchema methods" on page 521

GetSchema(String, String[]) method

Returns schema information for the data source of this "ULConnection class" on page 498 and, if specified, uses the specified string for the schema name and the specified string array for the restriction values.

Prototypes

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 on all columns in the database.
DataSourceInformation	Returns information about the database provider.
DataTypes	Returns a list of supported data types.

Collection	Metadata
ForeignKeys	Returns information on all foreign keys in the database.
IndexColumns	Returns information on all index columns in the database.
Indexes	Returns information on all indexes in the database.
MetaDataCollections	Returns a list of all collection names.
Publications	Returns information on all publications in the database.
ReservedWords	Returns a list of reserved words used by UltraLite.
Restrictions	Returns information on restrictions used in GetSchema.
Tables	Returns information on all tables in the database.

These collection names are also available as read-only properties in the "ULMetaDataCollectionNames class" on page 679.

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 on all tables named my_table.

GetSchema("Tables", new string[ ] { NULL, "User", NULL }) Returns information on all user tables.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "GetSchema methods" on page 521

GrantConnectTo method

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

Prototypes

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 498
- "ULConnection members" on page 498
- "UserID property" on page 539
- "Password property" on page 538
- "ConnectionString property" on page 503

Open method

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

Prototypes

```
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 498
- "ULConnection members" on page 498
- "ConnectionString property" on page 503
- "State property" on page 508

ResetLastDownloadTime method

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

Prototypes

Visual Basic

```
Public Sub ResetLastDownloadTime( _
  ByVal mask As Integer _
)
```

C#

```
public void ResetLastDownloadTime(
  int mask
);
```

Parameters

- **mask** The set of publications to reset. For more information, see the "ULPublicationSchema class" on page 720.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "GetLastDownloadTime method" on page 519

RevokeConnectFrom method

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

Prototypes

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 498
- "ULConnection members" on page 498
- "GrantConnectTo method" on page 523

RollbackPartialDownload method

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

Prototypes

Visual Basic
Public Sub **RollbackPartialDownload()**

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

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "KeepPartialDownload property" on page 800
- "ResumePartialDownload property" on page 803

StartSynchronizationDelete method

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

Prototypes

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 498
- "ULConnection members" on page 498
- "StopSynchronizationDelete method" on page 526
- "Truncate method" on page 848

StopSynchronizationDelete method

UL Ext.: Prevents delete operations from being synchronized.

Prototypes

Visual Basic
Public Sub **StopSynchronizationDelete()**

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

Remarks

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

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "StartSynchronizationDelete method" on page 526

Synchronize methods

UL Ext.: Synchronize the database using the current "SyncParms property" on page 509.

Synchronize() method

UL Ext.: Synchronize the database using the current "SyncParms property" on page 509.

Prototypes

Visual Basic
Public Sub **Synchronize()**

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

Remarks

A detailed result status is reported in this connection's "SyncResult property" on page 509.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "Synchronize methods" on page 527
- "Synchronize(ULSyncProgressListener) method" on page 527

Synchronize(ULSyncProgressListener) method

UL Ext.: Synchronize the database using the current "SyncParms property" on page 509 with progress events posted to the specified listener.

Prototypes

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

The last event posted to the listener will have a state of "ULSyncProgressState enumeration" on page 822.

Errors during synchronization will be posted as "ULSyncProgressState enumeration" on page 822 events and then thrown as "ULException class" on page 640s.

A detailed result status will be reported in this connection's "SyncResult property" on page 509.

See also

- "ULConnection class" on page 498
- "ULConnection members" on page 498
- "Synchronize methods" on page 527
- "ULSyncProgressListener interface" on page 820
- "Synchronize() method" on page 527

InfoMessage event

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

Prototypes

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" on page 678 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 498
- "ULConnection members" on page 498

StateChange event

Occurs when this connection changes state.

Prototypes

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

C#
public event override StateChangeEventHandler **StateChange** ;

Remarks

To process state change messages, you must create a 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 498
- "ULConnection members" on page 498

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.

Prototypes

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 ("Open method" on page 524) or dropping a database ("DropDatabase method" on page 588).

Leading and trailing spaces are ignored in filenames but respected in other values. Values that contain leading or trailing spaces, or a semicolon (;), or that begin with either a single quote (') or a double quote (") may not contain both single and double quotes.

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 "ToString method" on page 540. The resulting string is used to create a new "ULConnection class" on page 498 with the "ULConnection(String) constructor" on page 501 or set the "ConnectionString property" on page 503 of an existing "ULConnection class" on page 498 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 CE, 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 CE 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 informaton, see the "AdditionalParms property" on page 533.

Optional connection settings

Depending on your application's needs and how the database was created, you might need to supply a non-default "UserID property" on page 539 and "Password property" on page 538, a database "EncryptionKey property" on page 538, and the connection "CacheSize property" on page 536. If your application is using multiple connections, you should provide a unique "ConnectionName property" on page 536 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 "RevokeConnectFrom method" on page 525. To add a user or change a user's password, use the "GrantConnectTo method" on page 523.

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 "ChangeEncryptionKey method" on page 512.

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

See also

- "ULConnectionParms members" on page 532

ULConnectionParms members

Public constructors

Member name	Description
ULConnectionParms constructor	Initializes a ULConnectionParms instance with its default values.

Public properties

Member name	Description
AdditionalParms property	Specifies additional parameters as a semicolon-separated list of name=value pairs. These are less commonly used parameters.
CacheSize property	Specifies the size of the cache.
ConnectionName property	Specifies a name for the connection. This is only needed if you create more than one connection to the database.
DatabaseOnCE property	Specifies the path and filename of the UltraLite database on Windows CE.
DatabaseOnDesktop property	Specifies the path and filename of the UltraLite database on Windows desktop platforms.
EncryptionKey property	Specifies a key for encrypting the database.
Password property	Specifies the password for the authenticated user.

Member name	Description
UserID property	Specifies an authenticated user for the database.

Public methods

Member name	Description
ToString method	Returns the string representation of this instance.

Public events

Member name	Description
UnusedEvent event	Unused.

See also

- "ULConnectionParms class" on page 531

ULConnectionParms constructor

Initializes a ULConnectionParms instance with its default values.

Prototypes

Visual Basic
Public Sub **New()**

C#
public **ULConnectionParms();**

See also

- "ULConnectionParms class" on page 531
- "ULConnectionParms members" on page 532

AdditionalParms property

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

Prototypes

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 may need to be quoted as per the rules for "ConnectionString property" on page 503. 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 filename 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 filename parameter ("DatabaseOnCE property" on page 537 or "DatabaseOnDesktop property" on page 537). This parameter is required if the application (or UltraLite engine) needs to access two different databases that have the same base filename. This parameter is only used when opening a connection with "Open method" on page 524.

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, as well as 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 "Open method" on page 524.	
0	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. See the "RuntimeType property" on page 586 for more information on using the UltraLite engine with UltraLite.NET.

See also

- "ULConnectionParms class" on page 531
- "ULConnectionParms members" on page 532

CacheSize property

Specifies the size of the cache.

Prototypes

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"
```

The default cache size is 16 pages. Using the default page size of 4 KB, the default cache size is therefore 64 KB. The minimum cache size is platform dependent.

The default cache size is conservative. If your testing shows the need for better performance, you should increase the cache size.

Increasing the cache size beyond the size of the database itself provides no performance improvement and large cache sizes might interfere with the number of other applications you can use.

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

See also

- "ULConnectionParms class" on page 531
- "ULConnectionParms members" on page 532

ConnectionName property

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

Prototypes

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 531
- "ULConnectionParms members" on page 532

DatabaseOnCE property

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

Prototypes

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 531
- "ULConnectionParms members" on page 532

DatabaseOnDesktop property

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

Prototypes

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 531
- "ULConnectionParms members" on page 532

EncryptionKey property

Specifies a key for encrypting the database.

Prototypes

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 531
- "ULConnectionParms members" on page 532
- "ChangeEncryptionKey method" on page 512

Password property

Specifies the password for the authenticated user.

Prototypes

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 531
- "ULConnectionParms members" on page 532
- "UserID property" on page 539

UserID property

Specifies an authenticated user for the database.

Prototypes

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 "GrantConnectTo method" on page 523) and remove the DBA user (using "RevokeConnectFrom method" on page 525).

See also

- "ULConnectionParms class" on page 531
- "ULConnectionParms members" on page 532
- "Password property" on page 538
- "GrantConnectTo method" on page 523
- "RevokeConnectFrom method" on page 525

ToString method

Returns the string representation of this instance.

Prototypes

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 531
- "ULConnectionParms members" on page 532

UnusedEvent event

Unused.

Prototypes

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 531
- "ULConnectionParms members" on page 532

ULConnectionParms.UnusedEventHandler delegate

UL Ext.: Unused.

Prototypes

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.

Prototypes

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 ("Open method" on page 524) or dropping a database ("DropDatabase method" on page 588).

Leading and trailing spaces are ignored in filenames but respected in other values. Values that contain leading or trailing spaces, or a semicolon (;), or that begin with either a single quote (') or a double quote (") may not contain both single and double quotes.

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 DbConnectionStringBuilder.ConnectionString. The resulting string is used to create a new "ULConnection class" on page 498 with the "ULConnection(String) constructor" on page 501 or set the "ConnectionString property" on page 503 of an existing "ULConnection class" on page 498 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 CE, 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 CE 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 informaton, see the "DatabaseName property" on page 548.

Optional connection settings

Depending on your application's needs and how the database was created, you might need to supply a non-default "UserID property" on page 552 and "Password property" on page 550, a database "DatabaseKey property" on page 547, and the connection "CacheSize property" on page 546. If your application is using multiple connections, you should provide a unique "ConnectionName property" on page 547 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 "RevokeConnectFrom method" on page 525. To add a user or change a user's password, use the "GrantConnectTo method" on page 523.

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 "ChangeEncryptionKey method" on page 512.

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

See also

- "ULConnectionStringBuilder members" on page 543

ULConnectionStringBuilder members

Public constructors

Member name	Description
ULConnectionStringBuilder constructors	Initializes a new instance of the "ULConnectionStringBuilder class" on page 542.

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	**UL Ext.:** Specifies the size of the cache.
ConnectionName property	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.

Member name	Description
Count (inherited from DbConnectionStringBuilder)	Gets the current number of keys that are contained within the DbConnectionStringBuilder.ConnectionString.
DatabaseKey property	Specifies a key for encrypting the database.
DatabaseName property	Specifies a name for the database or the name of a loaded database to which a connection needs to be made.
DatabaseOnCE property	**UL Ext.:** Specifies the path and filename of the UltraLite database on Windows CE.
DatabaseOnDesktop property	**UL Ext.:** Specifies the path and filename 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	Specifies the value of the specified connection keyword.
Keys (inherited from DbConnectionStringBuilder)	Gets an ICollection that contains the keys in the DbConnectionStringBuilder.
Password property	Specifies the password for the authenticated user.
ReserveSize property	**UL Ext.:** Specifies the reserve file system space for storage of UltraLite persistent data.
StartLine property	Specifies the location and then starts the UltraLite engine.
UserID property	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	Determines whether the ULConnectionStringBuilder object contains a specific keyword.

Member name	Description
EquivalentTo method	Compares the connection information in this ULConnectionStringBuilder object with the connection information in the supplied DbConnectionStringBuilder object.
Remove method	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 (inherited from DbConnectionStringBuilder)	Retrieves a value corresponding to the supplied key from this DbConnectionStringBuilder.

See also

- "ULConnectionStringBuilder class" on page 542

ULConnectionStringBuilder constructors

Initializes a new instance of the "ULConnectionStringBuilder class" on page 542.

ULConnectionStringBuilder() constructor

Initializes a ULConnectionStringBuilder instance with its default values.

Prototypes

Visual Basic
Public Sub **New()**

C#
public **ULConnectionStringBuilder();**

See also

- "ULConnectionStringBuilder class" on page 542
- "ULConnectionStringBuilder members" on page 543
- "ULConnectionStringBuilder constructors" on page 545

ULConnectionStringBuilder(String) constructor

Initializes a ULConnectionStringBuilder instance with the specified connection string.

Prototypes

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 542
- "ULConnectionStringBuilder members" on page 543
- "ULConnectionStringBuilder constructors" on page 545

CacheSize property

UL Ext.: Specifies the size of the cache.

Prototypes

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"
```

The default cache size is 16 pages. Using the default page size of 4 KB, the default cache size is therefore 64 KB. The minimum cache size is platform dependent.

The default cache size is conservative. If your testing shows the need for better performance, you should increase the cache size.

Increasing the cache size beyond the size of the database itself provides no performance improvement and large cache sizes might interfere with the number of other applications you can use.

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

See also

- "ULConnectionStringBuilder class" on page 542
- "ULConnectionStringBuilder members" on page 543

ConnectionName property

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

Prototypes

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 542
- "ULConnectionStringBuilder members" on page 543

DatabaseKey property

Specifies a key for encrypting the database.

Prototypes

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 542
- "ULConnectionStringBuilder members" on page 543
- "ChangeEncryptionKey method" on page 512

DatabaseName property

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

Prototypes

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 filename 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 filename parameter ("DatabaseOnCE property" on page 548 or "DatabaseOnDesktop property" on page 549).

This parameter is required if the application (or UltraLite engine) needs to access two different databases that have the same base filename.

See also

- "ULConnectionStringBuilder class" on page 542
- "ULConnectionStringBuilder members" on page 543

DatabaseOnCE property

UL Ext.: Specifies the path and filename of the UltraLite database on Windows CE.

Prototypes

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 542
- "ULConnectionStringBuilder members" on page 543

DatabaseOnDesktop property

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

Prototypes

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 542
- "ULConnectionStringBuilder members" on page 543

Item property

Specifies the value of the specified connection keyword.

Prototypes

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 546
ce_file	"DatabaseOnCE property" on page 548
con	"ConnectionName property" on page 547
dbkey	"DatabaseKey property" on page 547
dbn	"DatabaseName property" on page 548
nt_file	"DatabaseOnDesktop property" on page 549
pwd	"Password property" on page 550
reserve_size	"ReserveSize property" on page 551
start	"StartLine property" on page 552
uid	"UserID property" on page 552

See also

- "ULConnectionStringBuilder class" on page 542
- "ULConnectionStringBuilder members" on page 543

Password property

Specifies the password for the authenticated user.

Prototypes

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 542
- "ULConnectionStringBuilder members" on page 543
- "UserID property" on page 552

ReserveSize property

UL Ext.: Specifies the reserve file system space for storage of UltraLite persistent data.

Prototypes

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, as well as 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 542
- "ULConnectionStringBuilder members" on page 543

StartLine property

Specifies the location and then starts the UltraLite engine.

Prototypes

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.

See the "RuntimeType property" on page 586 for more information on using the UltraLite engine with UltraLite.NET.

See also

- "ULConnectionStringBuilder class" on page 542
- "ULConnectionStringBuilder members" on page 543
- "RuntimeType property" on page 586

UserID property

Specifies an authenticated user for the database.

Prototypes

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 "GrantConnectTo method" on page 523) and remove the DBA user (using "RevokeConnectFrom method" on page 525).

See also

- "ULConnectionStringBuilder class" on page 542
- "ULConnectionStringBuilder members" on page 543
- "Password property" on page 550
- "GrantConnectTo method" on page 523
- "RevokeConnectFrom method" on page 525

ContainsKey method

Determines whether the ULConnectionStringBuilder object contains a specific keyword.

Prototypes

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 542
- "ULConnectionStringBuilder members" on page 543

EquivalentTo method

Compares the connection information in this ULConnectionStringBuilder object with the connection information in the supplied DbConnectionStringBuilder object.

Prototypes

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 542
- "ULConnectionStringBuilder members" on page 543

Remove method

Removes the entry with the specified key from the ULConnectionStringBuilder instance.

Prototypes

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 542
- "ULConnectionStringBuilder members" on page 543

ULCreateParms class

UL Ext.: Builds a string of creation-time options for creating an UltraLite database.

Prototypes

Visual Basic
Public Class **ULCreateParms**

C#
public class **ULCreateParms**

Remarks

A ULCreateParms object is used to specify the parameters for creating a database ("CreateDatabase method" on page 587).

Leading and trailing spaces are ignored in all string values. Values that contain a semicolon (;), or that begin with either a single quote (') or a double quote (") may not contain both single and double quotes.

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 "ToString method" on page 564. The resulting string can then be used as the createParms parameter of the "CreateDatabase method" on page 587.

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

Example

The following code creates the database \UltraLite\MyDatabase.udb on a Windows CE 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 556
- "GetDatabaseProperty method" on page 595

ULCreateParms members

Public constructors

Member name	Description
ULCreateParms constructor	Initializes a ULCreateParms instance with its default values.

Public properties

Member name	Description
CaseSensitive property	Specifies whether the new database should be case sensitive when comparing string values.
ChecksumLevel property	Specifies the level of database page checksums enabled for the new database.
DateFormat property	Specifies the date format used for string conversions by the new database.
DateOrder property	Specifies the date order used for string conversions by the new database.
FIPS property	Specifies whether the new database should be using AES_FIPS encryption or AES encryption.
MaxHashSize property	Specifies the default maximum number of bytes to use for index hashing in the new database.
NearestCentury property	Specifies the nearest century used for string conversions by the new database.
Obfuscate property	Specifies whether the new database should be using obfuscation (simple encryption) or not.
PageSize property	Specifies the page size of the new database, in bytes or kilobytes.
Precision property	Specifies the floating-point precision used for string conversions by the new database.

Member name	Description
Scale property	Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION during string conversions by the new database.
TimeFormat property	Specifies the time format used for string conversions by the new database.
TimestampFormat property	Specifies the timestamp format used for string conversions by the new database.
TimestampIncrement property	Specifies the minimum difference between two unique timestamps, in nanoseconds (1,000,000th of a second).
UTF8Encoding property	Specifies whether the new database should be using the UTF8 character set or the character set associated with the collation.

Public methods

Member name	Description
ToString method	Returns the string representation of this instance.

See also

- "ULCreateParms class" on page 555
- "GetDatabaseProperty method" on page 595

ULCreateParms constructor

Initializes a ULCreateParms instance with its default values.

Prototypes

Visual Basic
Public Sub **New()**

C#
public **ULCreateParms();**

See also

- "ULCreateParms class" on page 555
- "ULCreateParms members" on page 556

CaseSensitive property

Specifies whether the new database should be case sensitive when comparing string values.

Prototypes

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 555
- "ULCreateParms members" on page 556

ChecksumLevel property

Specifies the level of database page checksums enabled for the new database.

Prototypes

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 555
- "ULCreateParms members" on page 556

DateFormat property

Specifies the date format used for string conversions by the new database.

Prototypes

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 555
- "ULCreateParms members" on page 556

DateOrder property

Specifies the date order used for string conversions by the new database.

Prototypes

Visual Basic
Public Property **DateOrder** As ULDateOrder

C#
public ULDateOrder **DateOrder** { get; set; }

Property value

A "ULDateOrder enumeration" on page 636 value identifying the date order for string conversions. The default is YMD.

See also

- "ULCreateParms class" on page 555
- "ULCreateParms members" on page 556

FIPS property

Specifies whether the new database should be using AES_FIPS encryption or AES encryption.

Prototypes

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 (see the "EncryptionKey property" on page 538) when the new database is created. If FIPS is set true and no encryption key is supplied, the "CreateDatabase method" on page 587 will fail with a missing encryption key error.

See also

- "ULCreateParms class" on page 555
- "ULCreateParms members" on page 556

MaxHashSize property

Specifies the default maximum number of bytes to use for index hashing in the new database.

Prototypes

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 555
- "ULCreateParms members" on page 556

NearestCentury property

Specifies the nearest century used for string conversions by the new database.

Prototypes

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 555
- "ULCreateParms members" on page 556

Obfuscate property

Specifies whether the new database should be using obfuscation (simple encryption) or not.

Prototypes

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 ("FIPS property" on page 559). 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 555
- "ULCreateParms members" on page 556

PageSize property

Specifies the page size of the new database, in bytes or kilobytes.

Prototypes

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 555
- "ULCreateParms members" on page 556

Precision property

Specifies the floating-point precision used for string conversions by the new database.

Prototypes

Visual Basic
Public Property **Precision** As Integer

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

Property value

An integer specifying the precison. The value must be in the range [1,127]. The default is 30.

See also

- "ULCreateParms class" on page 555
- "ULCreateParms members" on page 556
- "Scale property" on page 562

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.

Prototypes

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 555
- "ULCreateParms members" on page 556

TimeFormat property

Specifies the time format used for string conversions by the new database.

Prototypes

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 555
- "ULCreateParms members" on page 556

TimestampFormat property

Specifies the timestamp format used for string conversions by the new database.

Prototypes

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 555
- "ULCreateParms members" on page 556

TimestampIncrement property

Specifies the minimum difference between two unique timestamps, in nanoseconds (1,000,000th of a second).

Prototypes

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 555
- "ULCreateParms members" on page 556

UTF8Encoding property

Specifies whether the new database should be using the UTF8 character set or the character set associated with the collation.

Prototypes

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 wish 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 555
- "ULCreateParms members" on page 556

ToString method

Returns the string representation of this instance.

Prototypes

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 555
- “ULCreateParms members” on page 556

ULCursorSchema class

UL Ext.: Represents the schema of an UltraLite.NET cursor. This class is abstract and so cannot be instantiated.

Prototypes

Visual Basic
MustInherit Public Class **ULCursorSchema**

C#
public abstract class **ULCursorSchema**

Remarks

This class is an abstract base class of the "ULTableSchema class" on page 849 and the "ULResultSetSchema class" on page 745.

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 566

ULCursorSchema members

Public properties

Member name	Description
ColumnCount property	Returns the number of columns in the cursor.
IsOpen property	Checks whether the cursor schema is currently open.
Name property	Returns the name of the cursor.

Public methods

Member name	Description
GetColumnID method	Returns the column ID of the named column.
GetColumnName method	Returns the name of the column identified by the specified column ID.
GetColumnPrecision method	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
GetColumnSQLName method	Returns the name of the column identified by the specified column ID.

Member name	Description
GetColumnScale method	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
GetColumnSize method	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
GetColumnULDbType method	Returns the UltraLite.NET data type of the column identified by the specified column ID.
GetSchemaTable method	Returns a DataTable that describes the column schema of the "ULDataReader class" on page 602.

See also

- "ULCursorSchema class" on page 566

ColumnCount property

Returns the number of columns in the cursor.

Prototypes

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 566
- "ULCursorSchema members" on page 566

IsOpen property

Checks whether the cursor schema is currently open.

Prototypes

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 566
- "ULCursorSchema members" on page 566

Name property

Returns the name of the cursor.

Prototypes

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 566
- "ULCursorSchema members" on page 566

GetColumnID method

Returns the column ID of the named column.

Prototypes

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 property" on page 567-1, 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 566
- "ULCursorSchema members" on page 566
- "ColumnCount property" on page 567

GetColumnName method

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

Prototypes

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 property" on page 567-1].

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 566
- "ULCursorSchema members" on page 566
- "ColumnCount property" on page 567

GetColumnPrecision method

Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).

Prototypes

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 property" on page 567-1].

Return value

The precision of the specified numeric column.

See also

- "ULCursorSchema class" on page 566
- "ULCursorSchema members" on page 566
- "GetColumnULDbType method" on page 572

GetColumnSQLName method

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

Prototypes

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 property" on page 567-1].

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 method" on page 569 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 likey 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 566
- "ULCursorSchema members" on page 566
- "ColumnCount property" on page 567

GetColumnScale method

Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).

Prototypes

```
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 property" on page 567-1].

Return value

The scale of the specified numeric column.

See also

- "ULCursorSchema class" on page 566
- "ULCursorSchema members" on page 566
- "GetColumnULDbType method" on page 572

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).

Prototypes

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 property" on page 567-1].

Return value

The size of the specified sized column.

See also

- "ULCursorSchema class" on page 566
- "ULCursorSchema members" on page 566
- "GetColumnULDbType method" on page 572

GetColumnULDbType method

Returns the UltraLite.NET data type of the column identified by the specified column ID.

Prototypes

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 property" on page 567-1].

Return value

A "ULDbType enumeration" on page 637 enumerated integer.

See also

- "ULCursorSchema class" on page 566
- "ULCursorSchema members" on page 566
- "ColumnCount property" on page 567

GetSchemaTable method

Returns a DataTable that describes the column schema of the "ULDataReader class" on page 602.

Prototypes

Visual Basic
```
Public Function GetSchemaTable() As DataTable
```

C#
```
public DataTable GetSchemaTable();
```

Return value

A DataTable that describes the column schema.

Remarks

For more information, see "GetSchemaTable method" on page 624.

See also

- "ULCursorSchema class" on page 566
- "ULCursorSchema members" on page 566

ULDataAdapter class

Represents a set of commands and a database connection used to fill a DataSet and to update a database. This class cannot be inherited.

Prototypes

Visual Basic
Public NotInheritable Class **ULDataAdapter**
  Inherits DbDataAdapter

C#
public sealed class **ULDataAdapter** : DbDataAdapter

Remarks

The DataSet provides a way to work with data offline; that is, away from your UltraLite database. The ULDataAdapter provides methods to associate a 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 class" on page 602 or the "ULTable class" on page 830 provide more efficient access to data.

Inherits: DbDataAdapter

Implements: IDbDataAdapter, IDataAdapter, IDisposable

See also

- "ULDataAdapter members" on page 574

ULDataAdapter members

Public constructors

Member name	Description
ULDataAdapter constructors	Initializes a new instance of the "ULDataAdapter class" on page 574.

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.

Member name	Description
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	Specifies a "ULCommand class" on page 462 object that is executed against the database when DbDataAdapter.Update is called to delete rows in the database that correspond to deleted rows in the DataSet.
FillLoadOption (inherited from DataAdapter)	Gets or sets the LoadOption that determines how the adapter fills the DataTable from the DbDataReader.
InsertCommand property	Specifies a "ULCommand class" on page 462 object that is executed against the database when DbDataAdapter.Update is called to insert rows in the database that correspond to inserted rows in the 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	Specifies a "ULCommand class" on page 462 that is used during DbDataAdapter.Fill or DbDataAdapter.FillSchema to obtain a result set from the database for copying into a DataSet.
TableMappings property	Returns a collection that provides the master mapping between a source table and a 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.
UpdateCommand property	Specifies a "ULCommand class" on page 462 object that is executed against the database when DbDataAdapter.Update is called to update rows in the database that correspond to updated rows in the 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	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.

Member name	Description
ShouldSerializeAcceptChanges-DuringFill (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 Db-DataAdapter)	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	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	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 574

ULDataAdapter constructors

Initializes a new instance of the "ULDataAdapter class" on page 574.

ULDataAdapter() constructor

Initializes a ULDataAdapter object.

Prototypes

Visual Basic
Public Sub **New()**

C#
public **ULDataAdapter();**

See also

- "ULDataAdapter class" on page 574
- "ULDataAdapter members" on page 574
- "ULDataAdapter constructors" on page 576
- "ULDataAdapter(ULCommand) constructor" on page 577

- "ULDataAdapter(String, ULConnection) constructor" on page 577
- "ULDataAdapter(String, String) constructor" on page 578

ULDataAdapter(ULCommand) constructor

Initializes a ULDataAdapter object with the specified SELECT statement.

Prototypes

Visual Basic

```
Public Sub New( _
  ByVal selectCommand As ULCommand _
)
```

C#

```
public  ULDataAdapter(
  ULCommand selectCommand
);
```

Parameters

- **selectCommand** A "ULCommand class" on page 462 object that is used during DbDataAdapter.Fill to select records from the data source for placement in the DataSet.

See also

- "ULDataAdapter class" on page 574
- "ULDataAdapter members" on page 574
- "ULDataAdapter constructors" on page 576
- "ULDataAdapter() constructor" on page 576
- "ULDataAdapter(String, ULConnection) constructor" on page 577
- "ULDataAdapter(String, String) constructor" on page 578

ULDataAdapter(String, ULConnection) constructor

Initializes a ULDataAdapter object with the specified SELECT statement and connection.

Prototypes

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 "SelectCommand property" on page 580 of the ULDataAdapter.
- **selectConnection** A "ULConnection class" on page 498 object that defines a connection to a database.

See also

- "ULDataAdapter class" on page 574
- "ULDataAdapter members" on page 574
- "ULDataAdapter constructors" on page 576
- "ULDataAdapter() constructor" on page 576
- "ULDataAdapter(ULCommand) constructor" on page 577
- "ULDataAdapter(String, String) constructor" on page 578

ULDataAdapter(String, String) constructor

Initializes a ULDataAdapter object with the specified SELECT statement and connection string.

Prototypes

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 "SelectCommand property" on page 580 of the ULDataAdapter.
- **selectConnectionString** A connection string for an UltraLite.NET database.

See also

- "ULDataAdapter class" on page 574
- "ULDataAdapter members" on page 574
- "ULDataAdapter constructors" on page 576
- "ULDataAdapter() constructor" on page 576
- "ULDataAdapter(ULCommand) constructor" on page 577
- "ULDataAdapter(String, ULConnection) constructor" on page 577

DeleteCommand property

Specifies a "ULCommand class" on page 462 object that is executed against the database when DbDataAdapter.Update is called to delete rows in the database that correspond to deleted rows in the DataSet.

Prototypes

Visual Basic
Public Property **DeleteCommand** As ULCommand

C#
public ULCommand **DeleteCommand** { get; set; }

Property value

A "ULCommand class" on page 462 object that is executed to delete rows in the database that correspond to deleted rows in the DataSet.

Remarks

When DeleteCommand is assigned to an existing "ULCommand class" on page 462 object, the "ULCommand class" on page 462 object is not cloned. The DeleteCommand maintains a reference to the existing "ULCommand class" on page 462.

This is the strongly-typed version of IDbDataAdapter.DeleteCommand and DbDataAdapter.DeleteCommand.

See also

- "ULDataAdapter class" on page 574
- "ULDataAdapter members" on page 574

InsertCommand property

Specifies a "ULCommand class" on page 462 object that is executed against the database when DbDataAdapter.Update is called to insert rows in the database that correspond to inserted rows in the DataSet.

Prototypes

Visual Basic
Public Property **InsertCommand** As ULCommand

C#
public ULCommand **InsertCommand** { get; set; }

Property value

A "ULCommand class" on page 462 object that is executed to insert rows in the database that correspond to inserted rows in the DataSet.

Remarks

When InsertCommand is assigned to an existing "ULCommand class" on page 462 object, the "ULCommand class" on page 462 object is not cloned. The InsertCommand maintains a reference to the existing "ULCommand class" on page 462.

This is the strongly-typed version of IDbDataAdapter.InsertCommand and DbDataAdapter.InsertCommand.

See also

- "ULDataAdapter class" on page 574
- "ULDataAdapter members" on page 574

SelectCommand property

Specifies a "ULCommand class" on page 462 that is used during DbDataAdapter.Fill or DbDataAdapter.FillSchema to obtain a result set from the database for copying into a DataSet.

Prototypes

Visual Basic
Public Property **SelectCommand** As ULCommand

C#
public ULCommand **SelectCommand** { get; set; }

Property value

A "ULCommand class" on page 462 object that is executed to fill the DataSet.

Remarks

When SelectCommand is assigned to an existing "ULCommand class" on page 462 object, the "ULCommand class" on page 462 object is not cloned. The SelectCommand maintains a reference to the existing "ULCommand class" on page 462.

If the SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised. The SELECT statement can also be specified in the "ULDataAdapter(ULCommand) constructor" on page 577, "ULDataAdapter(String, ULConnection) constructor" on page 577, or "ULDataAdapter(String, String) constructor" on page 578s.

This is the strongly-typed version of IDbDataAdapter.SelectCommand and DbDataAdapter.SelectCommand.

See also

- "ULDataAdapter class" on page 574
- "ULDataAdapter members" on page 574

TableMappings property

Returns a collection that provides the master mapping between a source table and a DataTable

Prototypes

Visual Basic
Public Readonly Property **TableMappings** As DataTableMappingCollection

C#
public DataTableMappingCollection **TableMappings** { get;}

Property value

A collection of DataTableMapping objects providing the master mapping between source tables and DataTables. The default value is an empty collection.

Remarks

When reconciling changes, the ULDataAdapter uses the DataTableMappingCollection collection to associate the column names used by the data source with the column names used by the DataSet.

This is the strongly-typed version of IDataAdapter.TableMappings.

See also

- "ULDataAdapter class" on page 574
- "ULDataAdapter members" on page 574

UpdateCommand property

Specifies a "ULCommand class" on page 462 object that is executed against the database when DbDataAdapter.Update is called to update rows in the database that correspond to updated rows in the DataSet.

Prototypes

Visual Basic
Public Property **UpdateCommand** As ULCommand

C#
public ULCommand **UpdateCommand** { get; set; }

Property value

A "ULCommand class" on page 462 object that is executed to update rows in the database that correspond to updated rows in the DataSet.

Remarks

When UpdateCommand is assigned to an existing "ULCommand class" on page 462 object, the "ULCommand class" on page 462 object is not cloned. The UpdateCommand maintains a reference to the existing "ULCommand class" on page 462.

If execution of this command returns rows, these rows may be merged with the DataSet depending on how you set the "UpdatedRowSource property" on page 472 of the "ULCommand class" on page 462 object.

This is the strongly-typed version of IDbDataAdapter.UpdateCommand and DbDataAdapter.DeleteCommand.

See also

- "ULDataAdapter class" on page 574
- "ULDataAdapter members" on page 574

GetFillParameters method

Returns the parameters set by the user when executing a SELECT statement.

Prototypes

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

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

Return value

An array of "ULParameter class" on page 687 objects that contains the parameters set by the user.

Remarks

This is the strongly-typed version of DbDataAdapter.GetFillParameters.

See also

- "ULDataAdapter class" on page 574
- "ULDataAdapter members" on page 574

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.

Prototypes

Visual Basic
Public Event **RowUpdated** As ULRowUpdatedEventHandler

C#
public event ULRowUpdatedEventHandler **RowUpdated** ;

Remarks

To process row updated events, you must create a "ULRowUpdatedEventHandler delegate" on page 754 delegate and attach it to this event.

Event data

- **Command** Returns the "ULCommand class" on page 462 executed when DbDataAdapter.Update 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 574
- "ULDataAdapter members" on page 574

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.

Prototypes

Visual Basic

```
Public Event RowUpdating As ULRowUpdatingEventHandler
```

C#

```
public event ULRowUpdatingEventHandler RowUpdating ;
```

Remarks

To process row updating events, you must create a "ULRowUpdatingEventHandler delegate" on page 758 delegate and attach it to this event.

Event data

- **Command** Specifies the "ULCommand class" on page 462 to execute when performing the DbDataAdapter.Update.
- **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 574
- "ULDataAdapter members" on page 574

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.

Prototypes

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) "DatabaseManager property" on page 506.

To use the UltraLite Engine runtime of UltraLite.NET, set "RuntimeType property" on page 586 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 585

ULDatabaseManager members

Public properties

Member name	Description
RuntimeType property	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	Creates a new UltraLite database.

Member name	Description
DropDatabase method	Deletes the specified database. You cannot drop a database that has open connections.
SetActiveSyncListener method	Specifies the listener object used to process ActiveSync calls from the MobiLink provider for ActiveSync.
SetServerSyncListener method	Specifies the listener object used to process the specified server synchronization message.
SignalSyncIsComplete method	Signals the MobiLink provider for ActiveSync that an application has completed synchronization.

See also

- "ULDatabaseManager class" on page 585

RuntimeType property

Specifies the UltraLite.NET runtime type. The runtime type must be selected before using any other UltraLite.NET API.

Prototypes

Visual Basic
Public Shared Property **RuntimeType** As ULRuntimeType

C#
public const ULRuntimeType **RuntimeType** { get; set; }

Property value

A "ULRuntimeType enumeration" on page 759 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 585

- "ULDatabaseManager members" on page 585

CreateDatabase method

Creates a new UltraLite database.

Prototypes

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 the "ULConnectionParms class" on page 531.
- **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 the "ULCreateParms class" on page 555.

Remarks

To specify a collation, you must first include the appropriate collation data source file from the src\ulcollations\cs (for C# projects) or src\ulcollations\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 CE device then opens a connection to it.

```
' Visual Basic
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnCE = "\UltraLite\MyDatabase.udb"
' Assumes file src\ulcollations\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 src\ulcollations\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 585
- "ULDatabaseManager members" on page 585
- "Open method" on page 524

DropDatabase method

Deletes the specified database.

You cannot drop a database that has open connections.

Prototypes

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 the "ULConnectionParms class" on page 531.

Example

The following code creates the database \UltraLite\MyDatabase.udb on a Windows CE 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 585
- "ULDatabaseManager members" on page 585
- "Open method" on page 524

SetActiveSyncListener method

Specifies the listener object used to process ActiveSync calls from the MobiLink provider for ActiveSync.

Prototypes

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 interface" on page 430 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 method" on page 590 or "SetActiveSyncListener method" on page 589 with a different *appClassName* fail.

To remove the ActiveSync listener, call "SetActiveSyncListener method" on page 589 with a null reference (Nothing in Visual Basic) as the *listener* parameter.

To remove all listeners, call "SetServerSyncListener method" on page 590 with a null reference (Nothing in Visual Basic) for all parameters.

Applications should remove all listeners prior to exiting.

Example

Refer to the "ActiveSyncInvoked method" on page 430 documentation for an example of "SetActiveSyncListener method" on page 589.

See also

- "ULDatabaseManager class" on page 585
- "ULDatabaseManager members" on page 585

SetServerSyncListener method

Specifies the listener object used to process the specified server synchronization message.

Prototypes

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 interface" on page 760 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 method" on page 590 or "SetActiveSyncListener method" on page 589 with a different *appClassName* fail.

To remove the listener for a particular message, call "SetServerSyncListener method" on page 590 with a null reference (Nothing in Visual Basic) as the *listener* parameter.

To remove all listeners, call "SetServerSyncListener method" on page 590 with a null reference (Nothing in Visual Basic) for all parameters.

Applications should remove all listeners before exiting.

Example

See the "ServerSyncInvoked method" on page 760 documentation for an example of "SetServerSyncListener method" on page 590.

See also

- "ULDatabaseManager class" on page 585
- "ULDatabaseManager members" on page 585

SignalSyncIsComplete method

Signals the MobiLink provider for ActiveSync that an application has completed synchronization.

Prototypes

Visual Basic
Public Sub **SignalSyncIsComplete()**

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

Example

Refer to the "ActiveSyncInvoked method" on page 430 documentation for an example of "SignalSyncIsComplete method" on page 591.

See also

- "ULDatabaseManager class" on page 585
- "ULDatabaseManager members" on page 585

ULDatabaseSchema class

UL Ext.: Represents the schema of an UltraLite.NET database. This class cannot be inherited.

Prototypes

Visual Basic
Public NotInheritable Class **ULDatabaseSchema**

C#
public sealed class **ULDatabaseSchema**

Remarks

There is no constructor for this class. A "ULDatabaseSchema class" on page 592 object is attached to a connection as its "Schema property" on page 507 and is only valid while that connection is open.

See also

- "ULDatabaseSchema members" on page 592

ULDatabaseSchema members

Public properties

Member name	Description
CollationName property	The name of the database's collation sequence.
IsCaseSensitive property	Checks whether the database is case sensitive.
IsOpen property	Whether the database schema is open.
PublicationCount property	The number of publications in the database.
TableCount property	The number of tables in the database.

Public methods

Member name	Description
GetDatabaseProperty method	Returns the value of the specified database property.
GetPublicationName method	Returns the name of the publication identified by the specified publication ID. Publication IDs are not publication masks.
GetPublicationSchema method	Returns the publication schema corresponding to the named publication.
GetTableCountInPublications method	Returns the number of tables included in the specified publication mask.
GetTableName method	Returns the name of the table identified by the specified table ID.

Member name	Description
SetDatabaseOption method	Sets the value for the specified database option.

See also

- "ULDatabaseSchema class" on page 592

CollationName property

The name of the database's collation sequence.

Prototypes

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 592
- "ULDatabaseSchema members" on page 592
- "GetDatabaseProperty method" on page 595

IsCaseSensitive property

Checks whether the database is case sensitive.

Prototypes

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 "UserID property" on page 539 and "Password property" on page 538 are verified.

See also

- "ULDatabaseSchema class" on page 592
- "ULDatabaseSchema members" on page 592
- "GetDatabaseProperty method" on page 595

IsOpen property

Whether the database schema is open.

Prototypes

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 592
- "ULDatabaseSchema members" on page 592

PublicationCount property

The number of publications in the database.

Prototypes

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. Publication IDs are not publication masks.

Note: Publication IDs, masks, and counts may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached IDs, masks, and counts after a schema upgrade.

See also

- "ULDatabaseSchema class" on page 592
- "ULDatabaseSchema members" on page 592
- "GetPublicationName method" on page 597

TableCount property

The number of tables in the database.

Prototypes

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 592
- "ULDatabaseSchema members" on page 592

GetDatabaseProperty method

Returns the value of the specified database property.

Prototypes

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 also affects how a connection's "UserID property" on page 539 and "Password property" on page 538 are verified.
CharSet	The character set of the database.
ChecksumLevel	The level of database page checksums enabled for the database.
C	CollationName	The name of the database's collation sequence.
ConnCount	The number of connections to the database.
date_format	The date format used for string conversions by the database. This format is not necessarily the same as the one used by 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.
MaxHashSize	The default maximum number of bytes to use for index hashing. This property can be set on a per-index basis.
ml_remote_id	The value of the ml_remote_id option used for identifying the database during synchronization.
Name	The name of the database (DBN).

Property	Description	
nearest_century	The nearest century used for string conversions by the database.	
PageSize	The page size of the database, in bytes.	
precision	The floating-point precision used for string conversions by the database.	
scale	The minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION during string conversions by the database.	
time_format	The time format used for string conversions by the database. This format is not necessarily the same as the one used by TimeSpan.	
timestamp_format	The timestamp format used for string conversions by the database. This format is not necessarily the same as the one used by DateTime.	
time	timestamp_increment	The minimum difference between two unique timestamps, in nanoseconds (1,000,000th of a second).

See also

- "ULDatabaseSchema class" on page 592
- "ULDatabaseSchema members" on page 592
- "SetDatabaseOption method" on page 600
- "CollationName property" on page 593

GetPublicationName method

Returns the name of the publication identified by the specified publication ID. Publication IDs are not publication masks.

Prototypes

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 property" on page 594].

Return value

The publication name as a string.

Remarks

Note: Publication IDs, masks, and counts may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached IDs, masks, and counts after a schema upgrade.

See also

- "ULDatabaseSchema class" on page 592
- "ULDatabaseSchema members" on page 592
- "PublicationCount property" on page 594

GetPublicationSchema method

Returns the publication schema corresponding to the named publication.

Prototypes

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 class" on page 720 object representing the named publication.

See also

- "ULDatabaseSchema class" on page 592
- "ULDatabaseSchema members" on page 592
- "GetPublicationName method" on page 597
- "ULPublicationSchema class" on page 720

GetTableCountInPublications method

Returns the number of tables included in the specified publication mask.

Prototypes

Visual Basic

```
Public Function GetTableCountInPublications( _
  ByVal mask As Integer _
) As Integer
```

C#

```
public int GetTableCountInPublications(
  int mask
);
```

Parameters

- **mask** The set of publications to check. For information on building publication masks, see "ULPublicationSchema class" on page 720.

Return value

The number of tables included in set of publications.

Remarks

The count will not include tables whose names end in _nosync.

Note: Publication IDs, masks, and counts may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached IDs, masks, and counts after a schema upgrade.

See also

- "ULDatabaseSchema class" on page 592
- "ULDatabaseSchema members" on page 592

GetTableName method

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

Prototypes

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 592
- "ULDatabaseSchema members" on page 592
- "TableCount property" on page 595

SetDatabaseOption method

Sets the value for the specified database option.

Prototypes

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,UInt32.MaxValue]. The default is "INVALID_DATABASE_ID field" on page 502 (used to indicate that the database ID has not been set for the current database).
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 592
- "ULDatabaseSchema members" on page 592
- "GetDatabaseProperty method" on page 595

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.

Prototypes

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 class" on page 462:

```
' 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 "ExecuteResultSet() method" on page 486, "ExecuteTable() method" on page 489, or a "ULDataAdapter class" on page 574. The ULDataReader retrieves rows as needed, whereas the "ULDataAdapter class" on page 574 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 method" on page 626.

Inherits: DbDataReader

Implements: IDataReader, IDataRecord, IDisposable

See also

- "ULDataReader members" on page 603

ULDataReader members

Public properties

Member name	Description
Depth property	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
FieldCount property	Returns the number of columns in the cursor.
HasRows property	Checks whether the ULDataReader has one or more rows.
IsBOF property	**UL Ext.:** Checks whether the current row position is before the first row.
IsClosed property	Checks whether the cursor is currently open.
IsEOF property	**UL Ext.:** Checks whether the current row position is after the last row.
Item properties	Gets the value of a specified column as an instance of Object.
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.
RowCount property	**UL Ext.:** Returns the number of rows in the cursor.
Schema property	**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
Close method	Closes the cursor.
Dispose (inherited from DbDataReader)	Releases the resources consumed by this DbDataReader.
GetBoolean method	Returns the value for the specified column as a Boolean.
GetByte method	Returns the value for the specified column as an unsigned 8-bit value (Byte).
GetBytes methods	**UL Ext.:** Returns the value for the specified column as an array of Bytes. Only valid for columns of type "ULDbType enumeration" on page 637, "ULDbType enumeration" on page 637, or "ULDbType enumeration" on page 637.
GetChar method	This method is not supported in UltraLite.NET.

Member name	Description
GetChars method	Copies a subset of the value for the specified "ULDbType enumeration" on page 637 column, beginning at the specified offset, to the specified offset of the destination Char array.
GetData (inherited from DbDataReader)	Returns a DbDataReader object for the requested column ordinal.
GetDataTypeName method	Returns the name of the specified column's provider data type.
GetDateTime method	Returns the value for the specified column as a DateTime with millisecond accuracy.
GetDecimal method	Returns the value for the specified column as a Decimal.
GetDouble method	Returns the value for the specified column as a Double.
GetEnumerator method	Returns an IEnumerator that iterates through the ULDataReader.
GetFieldType method	Returns the Type most appropriate for the specified column.
GetFloat method	Returns the value for the specified column as a Single.
GetGuid method	Returns the value for the specified column as a UUID (Guid).
GetInt16 method	Returns the value for the specified column as an Int16.
GetInt32 method	Returns the value for the specified column as an Int32.
GetInt64 method	Returns the value for the specified column as an Int64.
GetName method	Returns the name of the specified column.
GetOrdinal method	Returns the column ID of the named column.
GetProviderSpecificFieldType (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	Returns a DataTable that describes the column metadata of the ULDataReader.
GetString method	Returns the value for the specified column as a String.
GetTimeSpan method	Returns the value for the specified column as a TimeSpan with millisecond accuracy.
GetUInt16 method	Returns the value for the specified column as a UInt16.

Member name	Description
GetUInt32 method	Returns the value for the specified column as a UInt32.
GetUInt64 method	Returns the value for the specified column as a UInt64.
GetValue method	Returns the value of the specified column in its native format.
GetValues method	Returns all the column values for the current row.
IsDBNull method	Checks whether the value from the specified column is NULL.
MoveAfterLast method	**UL Ext.:** Positions the cursor to after the last row of the cursor.
MoveBeforeFirst method	**UL Ext.:** Positions the cursor to before the first row of the cursor.
MoveFirst method	**UL Ext.:** Positions the cursor to the first row of the cursor.
MoveLast method	**UL Ext.:** Positions the cursor to the last row of the cursor.
MoveNext method	**UL Ext.:** Positions the cursor to the next row or after the last row if the cursor was already on the last row.
MovePrevious method	**UL Ext.:** Positions the cursor to the previous row or before the first row.
MoveRelative method	**UL Ext.:** Positions the cursor relative to the current row.
NextResult method	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
Read method	Positions the cursor to the next row, or after the last row if the cursor was already on the last row.

See also

- "ULDataReader class" on page 602

Depth property

Returns the depth of nesting for the current row. The outermost table has a depth of zero.

Prototypes

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 602
- "ULDataReader members" on page 603

FieldCount property

Returns the number of columns in the cursor.

Prototypes

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 "ColumnCount property" on page 567.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

HasRows property

Checks whether the ULDataReader has one or more rows.

Prototypes

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 602
- "ULDataReader members" on page 603

IsBOF property

UL Ext.: Checks whether the current row position is before the first row.

Prototypes

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 602
- "ULDataReader members" on page 603

IsClosed property

Checks whether the cursor is currently open.

Prototypes

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 602
- "ULDataReader members" on page 603

IsEOF property

UL Ext.: Checks whether the current row position is after the last row.

Prototypes

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 602
- "ULDataReader members" on page 603

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.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "GetValue method" on page 629.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "Item properties" on page 608
- "GetFieldType method" on page 619
- "Item(String) property" on page 609

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.

Prototypes

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 602
- "ULDataReader members" on page 603
- "Item properties" on page 608
- "Item(Int32) property" on page 608
- "GetOrdinal method" on page 624
- "GetValue method" on page 629
- "GetFieldType method" on page 619

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.

Prototypes

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 602
- "ULDataReader members" on page 603

RowCount property

UL Ext.: Returns the number of rows in the cursor.

Prototypes

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 "StopSynchronizationDelete method" on page 526.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "StartSynchronizationDelete method" on page 526
- "StopSynchronizationDelete method" on page 526

Schema property

UL Ext.: Holds the schema of this cursor.

Prototypes

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

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

Property value

For result sets, the "ULResultSetSchema class" on page 745 object representing the schema of the result set. For tables, the "ULTableSchema class" on page 849 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 "GetSchemaTable method" on page 624.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

Close method

Closes the cursor.

Prototypes

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 602
- "ULDataReader members" on page 603

GetBoolean method

Returns the value for the specified column as a Boolean.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a Boolean.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetByte method

Returns the value for the specified column as an unsigned 8-bit value (Byte).

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a Byte.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetBytes methods

UL Ext.: Returns the value for the specified column as an array of Bytes. Only valid for columns of type "ULDbType enumeration" on page 637, "ULDbType enumeration" on page 637, or "ULDbType enumeration" on page 637.

GetBytes(Int32, Int64, Byte[], Int32, Int32) method

Copies a subset of the value for the specified "ULDbType enumeration" on page 637 column, beginning at the specified offset, to the specified offset of the destination Byte array.

Prototypes

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,"FieldCount property" on page 606-1]. 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 class" on page 640 with code "ULSQLCode enumeration" on page 763 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 class" on page 640 with the appropriate error code is thrown.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetBytes methods" on page 612
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619
- "GetBytes(Int32) method" on page 614

GetBytes(Int32) method

UL Ext.: Returns the value for the specified column as an array of Bytes. Only valid for columns of type "ULDbType enumeration" on page 637, "ULDbType enumeration" on page 637, or "ULDbType enumeration" on page 637.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as an array of Bytes.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetBytes methods" on page 612
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619
- "GetBytes(Int32, Int64, Byte[], Int32, Int32) method" on page 613

GetChar method

This method is not supported in UltraLite.NET.

Prototypes

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,"FieldCount property" on page 606-1]. 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 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619
- "GetString method" on page 626

GetChars method

Copies a subset of the value for the specified "ULDbType enumeration" on page 637 column, beginning at the specified offset, to the specified offset of the destination Char array.

Prototypes

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,"FieldCount property" on page 606-1]. 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 class" on page 640 with code "ULSQLCode enumeration" on page 763 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 class" on page 640 with the appropriate error code is thrown.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetDataTypeName method

Returns the name of the specified column's provider data type.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

A string corresponding to the column's "ULDbType enumeration" on page 637.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetColumnULDbType method" on page 572

GetDateTime method

Returns the value for the specified column as a DateTime with millisecond accuracy.

Prototypes

```
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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a DateTime.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetDecimal method

Returns the value for the specified column as a Decimal.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a Decimal.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetDouble method

Returns the value for the specified column as a Double.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a Double.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetEnumerator method

Returns an IEnumerator that iterates through the ULDataReader.

Prototypes

Visual Basic
Public Overrides Function **GetEnumerator()** As IEnumerator

C#
public override IEnumerator **GetEnumerator();**

Return value

A IEnumerator for the ULDataReader.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

GetFieldType method

Returns the Type most appropriate for the specified column.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

A Type value for the column.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetDataTypeName method" on page 616
- "GetColumnULDbType method" on page 572

GetFloat method

Returns the value for the specified column as a Single.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a Single.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetGuid method

Returns the value for the specified column as a UUID (Guid).

Prototypes

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,"FieldCount property" on page 606-1]. 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 enumeration" on page 637 or for columns of type "ULDbType enumeration" on page 637 with length 16.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619
- "GetColumnULDbType method" on page 572
- "GetColumnSize method" on page 572

GetInt16 method

Returns the value for the specified column as an Int16.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as an Int16.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetInt32 method

Returns the value for the specified column as an Int32.

Prototypes

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,"FieldCount property" on page 606-1]. 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 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetInt64 method

Returns the value for the specified column as an Int64.

Prototypes

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,"FieldCount property" on page 606-1]. 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 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetName method

Returns the name of the specified column.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "GetColumnName method" on page 569.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

- "FieldCount property" on page 606
- "GetSchemaTable method" on page 624

GetOrdinal method

Returns the column ID of the named column.

Prototypes

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 "FieldCount property" on page 606-1, 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 "GetColumnID method" on page 568.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetSchemaTable method" on page 624

GetSchemaTable method

Returns a DataTable that describes the column metadata of the ULDataReader.

Prototypes

Visual Basic
Public Overrides Function **GetSchemaTable()** As DataTable

C#
public override DataTable **GetSchemaTable();**

Return value

A 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 property" on page 606-1].
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 enumeration" on page 637 or "ULDbType enumeration" on page 637) or DBNull if the column is not numeric.
NumericScale	The scale of a numeric column (ProviderType "ULDbType enumeration" on page 637 or "ULDbType enumeration" on page 637) or DBNull if the column is not numeric.
IsUnique	True if the column is a non-computed unique column in the table (BaseTableName) it is taken from.
IsKey	True if the column is one of a set of columns in the result set that taken together from a unique key for the result set. The set of columns with IsKey set to true does not need to be the minimal set that uniquely identifies a row in the result set.
BaseCatalogName	The name of the catalog in the database that contains the column. For UltraLite.NET, this value is always DBNull.
BaseColumnName	The original name of the column in the table BaseTableName of the database or DBNull if the column is computed or if this information cannot be determined.
BaseSchemaName	The name of the schema in the database that contains the column. For UltraLite.NET, this value is always DBNull.

DataTable column	Description
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 enumeration" on page 637 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 enumeration" on page 637 or a "ULDbType enumeration" on page 637 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 602
- "ULDataReader members" on page 603
- "Schema property" on page 610

GetString method

Returns the value for the specified column as a String.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a String.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetTimeSpan method

Returns the value for the specified column as a TimeSpan with millisecond accuracy.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a TimeSpan.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetUInt16 method

Returns the value for the specified column as a UInt16.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as an UInt16.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetUInt32 method

Returns the value for the specified column as a UInt32.

Prototypes

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,"FieldCount property" on page 606-1]. 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 602

- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetUInt64 method

Returns the value for the specified column as a UInt64.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Return value

The column value as a UInt64

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetValue method

Returns the value of the specified column in its native format.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Item(Int32) property" on page 608.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

GetValues method

Returns all the column values for the current row.

Prototypes

Visual Basic

```
Public Overrides Function GetValues( _
  ByVal values As Object() _
) As Integer
```

C#

```
public override int GetValues(
  object[] values
);
```

Parameters

- **values** The array of 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 ("FieldCount property" on page 606), 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 Object array that contains fewer than the number of columns contained in the resulting row. Only the amount of data the Object array holds is copied to the array. You can also pass an 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 602
- "ULDataReader members" on page 603
- "FieldCount property" on page 606
- "GetFieldType method" on page 619
- "GetValue method" on page 629

IsDBNull method

Checks whether the value from the specified column is NULL.

Prototypes

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,"FieldCount property" on page 606-1]. 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 602
- "ULDataReader members" on page 603
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

MoveAfterLast method

UL Ext.: Positions the cursor to after the last row of the cursor.

Prototypes

Visual Basic

```
Public Sub MoveAfterLast()
```


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

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

MoveBeforeFirst method

UL Ext.: Positions the cursor to before the first row of the cursor.

Prototypes

Visual Basic
Public Sub **MoveBeforeFirst()**

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

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

MoveFirst method

UL Ext.: Positions the cursor to the first row of the cursor.

Prototypes

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 602
- "ULDataReader members" on page 603

MoveLast method

UL Ext.: Positions the cursor to the last row of the cursor.

Prototypes

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 602
- "ULDataReader members" on page 603

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.

Prototypes

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 "Read method" on page 635.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

MovePrevious method

UL Ext.: Positions the cursor to the previous row or before the first row.

Prototypes

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 602
- "ULDataReader members" on page 603

MoveRelative method

UL Ext.: Positions the cursor relative to the current row.

Prototypes

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 ("IsEOF property" on page 607) if

offset is positive, and before the first row ("IsBOF property" on page 607) if the

offset is negative.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

NextResult method

Advances the ULDataReader to the next result when reading the results of batch SQL statements.

Prototypes

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, hence the ULDataReader is always positioned on the first and only result set. Calling NextResult has no effect.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

Read method

Positions the cursor to the next row, or after the last row if the cursor was already on the last row.

Prototypes

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 "MoveNext method" on page 633.

See also

- "ULDataReader class" on page 602
- "ULDataReader members" on page 603

ULDateOrder enumeration

UL Ext.: Enumerates the date orders that a database can support.

Prototypes

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 559

ULDbType enumeration

Enumerates the UltraLite.NET database data types.

Prototypes

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.
UnsignedSmall-Int	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
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

Member name	Description	Value
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
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 619
- "GetDataTypeName method" on page 616
- "GetColumnULDbType method" on page 572

ULException class

Represents a SQL error returned by the UltraLite.NET database. This class cannot be inherited.

Prototypes

Visual Basic
Public NotInheritable Class **ULException**
  Inherits ApplicationException

C#
public sealed class **ULException** : ApplicationException

Remarks

The SQL code denoting the error is returned in the "NativeError property" on page 642.

This class is not serializable under the .NET Compact Framework.

See also

- "ULException members" on page 640

ULException members

Public constructors

Member name	Description
ULException constructor	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.
NativeError property	Returns the SQL code returned by the database.
Source property	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.

Member name	Description
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	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 640

ULException constructor

Creates a ULException with the given error code.

Prototypes

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 "ULSQLCode enumeration" on page 763 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 640
- "ULException members" on page 640

NativeError property

Returns the SQL code returned by the database.

Prototypes

Visual Basic
Public Readonly Property **NativeError** As ULSQLCode

C#
public ULSQLCode **NativeError** { get;}

Property value

The "ULSQLCode enumeration" on page 763 value returned by the database.

See also

- "ULException class" on page 640
- "ULException members" on page 640

Source property

Returns the name of the provider that generated the error.

Prototypes

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 640
- "ULException members" on page 640

GetObjectData method

Populates a SerializationInfo with the data needed to serialize this ULException.

Prototypes

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 640
- "ULException members" on page 640

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.

Prototypes

Visual Basic
Public NotInheritable Class **ULFactory**
  Inherits DbProviderFactory

C#
public sealed class **ULFactory** : DbProviderFactory

Remarks

ADO.NET 2.0 adds two new classes, the DbProviderFactories and the DbProviderFactory, 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: DbProviderFactory

See also

- "ULFactory members" on page 645

ULFactory members

Public fields

Member name	Description
Instance field	Represents the singleton instance of the ULFactory class. This field is read-only.

Public properties

Member name	Description
CanCreateDataSourceEnumerator property	Returns false to indicate the UltraLite.NET does not support DbDataSourceEnumerator.

Public methods

Member name	Description
CreateCommand method	Returns a strongly typed DbCommand instance.
CreateCommandBuilder (inherited from DbProviderFactory)	Returns a new instance of the provider's class that implements the DbCommandBuilder.
CreateConnection method	Returns a strongly typed DbConnection instance.
CreateConnectionStringBuilder method	Returns a strongly typed DbConnectionStringBuilder instance.
CreateDataAdapter method	Returns a strongly typed DbDataAdapter instance.
CreateDataSourceEnumerator (inherited from DbProviderFactory)	Returns a new instance of the provider's class that implements the DbDataSourceEnumerator.
CreateParameter method	Returns a strongly typed 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 644

Instance field

Represents the singleton instance of the ULFactory class. This field is read-only.

Prototypes

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 DbProviderFactories.GetFactory. For an example, see the "ULFactory class" on page 644 description.

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

See also

- "ULFactory class" on page 644
- "ULFactory members" on page 645

CanCreateDataSourceEnumerator property

Returns false to indicate the UltraLite.NET does not support DbDataSourceEnumerator.

Prototypes

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 644
- "ULFactory members" on page 645

CreateCommand method

Returns a strongly typed DbCommand instance.

Prototypes

Visual Basic
Public Overrides Function **CreateCommand()** As DbCommand

C#
public override DbCommand **CreateCommand();**

Return value

A new "ULCommand class" on page 462 instance typed as DbCommand.

See also

- "ULFactory class" on page 644
- "ULFactory members" on page 645

CreateConnection method

Returns a strongly typed DbConnection instance.

Prototypes

Visual Basic
Public Overrides Function **CreateConnection()** As DbConnection

C#
public override DbConnection **CreateConnection();**

Return value

A new "ULConnection class" on page 498 instance typed as DbConnection.

See also

- "ULFactory class" on page 644
- "ULFactory members" on page 645

CreateConnectionStringBuilder method

Returns a strongly typed DbConnectionStringBuilder instance.

Prototypes

Visual Basic
Public Overrides Function **CreateConnectionStringBuilder()** As DbConnectionStringBuilder

C#
public override DbConnectionstring Builder **CreateConnectionStringBuilder();**

Return value

A new "ULConnectionStringBuilder class" on page 542 instance typed as DbConnectionStringBuilder.

See also

- "ULFactory class" on page 644
- "ULFactory members" on page 645

CreateDataAdapter method

Returns a strongly typed DbDataAdapter instance.

Prototypes

Visual Basic
Public Overrides Function **CreateDataAdapter()** As DbDataAdapter

C#
public override DbDataAdapter **CreateDataAdapter();**

Return value

A new "ULDataAdapter class" on page 574 instance typed as DbDataAdapter.

See also

- "ULFactory class" on page 644
- "ULFactory members" on page 645

CreateParameter method

Returns a strongly typed DbParameter instance.

Prototypes

Visual Basic
Public Overrides Function **CreateParameter()** As DbParameter

C#
public override DbParameter **CreateParameter();**

Return value

A new "ULParameter class" on page 687 instance typed as DbParameter.

See also

- "ULFactory class" on page 644
- "ULFactory members" on page 645

ULFileTransfer class

UL Ext.: Transfers a file from a remote database using the MobiLink server. This class cannot be inherited.

Prototypes

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 "RuntimeType property" on page 586 to the appropriate value before using this API or any other UltraLite.NET API.

To transfer a file you must set the "FileName property" on page 654, "Stream property" on page 656, "UserName property" on page 658, and "Version property" on page 659.

See also

- "ULFileTransfer members" on page 649

ULFileTransfer members

Public constructors

Member name	Description
ULFileTransfer constructor	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	Returns the authorization status code for the last file transfer attempt.
AuthValue property	Returns the return value from custom user authentication synchronization scripts.
AuthenticationParms property	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
DestinationFileName property	Specifies the local file name for the downloaded file.
DestinationPath property	Specifies where to download the file.
DownloadedFile property	Checks whether the file was actually downloaded during the last file transfer attempt.

Member name	Description
FileAuthCode property	Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.
FileName property	Specifies the name of the file to download.
ForceDownload property	Specifies whether to force the download of the file if it exists.
Password property	The MobiLink password for the user specified by UserName.
ResumePartialDownload property	Specifies whether to resume or discard a previous partial download.
Stream property	Specifies the MobiLink synchronization stream to use for the file transfer.
StreamErrorCode property	Returns the error reported by the stream itself for the last file transfer attempt.
StreamErrorSystem property	Returns the stream error system-specific code.
StreamParms property	Specifies the parameters to configure the synchronization stream.
UserName property	The user name that uniquely identifies the MobiLink client to the MobiLink server.
Version property	Specifies which synchronization script to use.

Public methods

Member name	Description
DownloadFile methods	Download the file specified by the properties of this object.

See also

- "ULFileTransfer class" on page 649

ULFileTransfer constructor

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

Prototypes

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 "RuntimeType property" on page 586 to the appropriate value before using this API or any other UltraLite.NET API.

The ULFileTransfer object needs to have the "FileName property" on page 654, "Stream property" on page 656, "UserName property" on page 658, and "Version property" on page 659 set before it can transfer a file.

See also

- "ULFileTransfer class" on page 649
- "ULFileTransfer members" on page 649
- "DownloadFile() method" on page 660

AuthStatus property

Returns the authorization status code for the last file transfer attempt.

Prototypes

Visual Basic
Public Readonly Property **AuthStatus** As ULAuthStatusCode

C#
public ULAuthStatusCode **AuthStatus** { get;}

Property value

One of the "ULAuthStatusCode enumeration" on page 433 values denoting the authorization status for the last file transfer attempt.

See also

- "ULFileTransfer class" on page 649
- "ULFileTransfer members" on page 649

AuthValue property

Returns the return value from custom user authentication synchronization scripts.

Prototypes

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 649
- "ULFileTransfer members" on page 649

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Prototypes

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 649
- "ULFileTransfer members" on page 649

DestinationFileName property

Specifies the local file name for the downloaded file.

Prototypes

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 property" on page 654 is used. The default is a null reference (Nothing in Visual Basic).

See also

- "ULFileTransfer class" on page 649

- "ULFileTransfer members" on page 649

DestinationPath property

Specifies where to download the file.

Prototypes

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 649
- "ULFileTransfer members" on page 649
- "ForceDownload property" on page 655
- "DownloadFile() method" on page 660

DownloadedFile property

Checks whether the file was actually downloaded during the last file transfer attempt.

Prototypes

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() method" on page 660 is invoked, it will return true, but DownloadedFile will be false. If an error occurs and "DownloadFile() method" on page 660 returns false, DownloadedFile will be false.

See also

- "ULFileTransfer class" on page 649
- "ULFileTransfer members" on page 649

FileAuthCode property

Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.

Prototypes

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 649
- "ULFileTransfer members" on page 649

FileName property

Specifies the name of the file to download.

Prototypes

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 649
- "ULFileTransfer members" on page 649
- "DestinationFileName property" on page 652
- "DownloadFile() method" on page 660

ForceDownload property

Specifies whether to force the download of the file if it exists.

Prototypes

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 649
- "ULFileTransfer members" on page 649
- "ResumePartialDownload property" on page 656
- "DownloadFile() method" on page 660

Password property

The MobiLink password for the user specified by UserName.

Prototypes

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 649
- "ULFileTransfer members" on page 649
- "UserName property" on page 658
- "DownloadFile() method" on page 660

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Prototypes

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 649
- "ULFileTransfer members" on page 649
- "ForceDownload property" on page 655
- "DownloadFile() method" on page 660

Stream property

Specifies the MobiLink synchronization stream to use for the file transfer.

Prototypes

Visual Basic
Public Property **Stream** As ULStreamType

C#
public ULStreamType **Stream** { get; set; }

Property value

One of the "ULStreamType enumeration" on page 794 values specifying the type of synchronization stream to use. The default is "ULStreamType enumeration" on page 794.

Remarks

Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. These parameters are supplied by the "StreamParms property" on page 658.

If the stream type is set to a value that is invalid for the platform, the stream type is set to "ULStreamType enumeration" on page 794.

See also

- "ULFileTransfer class" on page 649
- "ULFileTransfer members" on page 649
- "ULStreamType enumeration" on page 794
- "StreamParms property" on page 658
- "DownloadFile() method" on page 660

StreamErrorCode property

Returns the error reported by the stream itself for the last file transfer attempt.

Prototypes

Visual Basic
Public Readonly Property **StreamErrorCode** As ULStreamErrorCode

C#
public ULStreamErrorCode **StreamErrorCode** { get;}

Property value

One of the "ULStreamErrorCode enumeration" on page 773 values denoting the error reported by the stream itself, "ULStreamErrorCode enumeration" on page 773 if no error occurred.

See also

- "ULFileTransfer class" on page 649
- "ULFileTransfer members" on page 649

StreamErrorSystem property

Returns the stream error system-specific code.

Prototypes

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 649
- "ULFileTransfer members" on page 649

StreamParms property

Specifies the parameters to configure the synchronization stream.

Prototypes

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 on configuring specific stream types, see "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*].

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 649
- "ULFileTransfer members" on page 649
- "Stream property" on page 656
- "DownloadFile() method" on page 660
- "ULStreamType enumeration" on page 794

UserName property

The user name that uniquely identifies the MobiLink client to the MobiLink server.

Prototypes

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 649
- "ULFileTransfer members" on page 649
- "Password property" on page 655
- "DownloadFile() method" on page 660

Version property

Specifies which synchronization script to use.

Prototypes

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 649
- "ULFileTransfer members" on page 649
- "DownloadFile() method" on page 660

DownloadFile methods

Download the file specified by the properties of this object.

DownloadFile() method

Download the file specified by the properties of this object.

Prototypes

Visual Basic
Public Function **DownloadFile()** As Boolean

C#
public bool **DownloadFile();**

Return value

True if successful, false otherwise (check "StreamErrorCode property" on page 657 and other status properties for reason).

Remarks

The file specified by the "FileName property" on page 654 is downloaded by the MobiLink server to the "DestinationPath property" on page 653 using the "Stream property" on page 656, "UserName property" on page 658, "Password property" on page 655, and "Version property" on page 659. Other properties that affect the download are "DestinationFileName property" on page 652, "AuthenticationParms property" on page 652, "ForceDownload property" on page 655 and "ResumePartialDownload property" on page 656.

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 "AuthStatus property" on page 651, "AuthValue property" on page 651, "FileAuthCode property" on page 654, "DownloadedFile property" on page 653, "StreamErrorCode property" on page 657, and "StreamErrorSystem property" on page 657.

See also

- "ULFileTransfer class" on page 649
- "ULFileTransfer members" on page 649
- "DownloadFile methods" on page 660
- "DownloadFile(ULFileTransferProgressListener) method" on page 660

DownloadFile(ULFileTransferProgressListener) method

Download the file specified by the properties of this object with progress events posted to the specified listener.

Prototypes

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 "StreamErrorCode property" on page 657 and other status properties for reason).

Remarks

The file specified by the "FileName property" on page 654 is downloaded by the MobiLink server to the "DestinationPath property" on page 653 using the "Stream property" on page 656, "UserName property" on page 658, "Password property" on page 655, and "Version property" on page 659. Other properties that affect the download are "DestinationFileName property" on page 652, "AuthenticationParms property" on page 652, "ForceDownload property" on page 655 and "ResumePartialDownload property" on page 656.

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 "AuthStatus property" on page 651, "AuthValue property" on page 651, "FileAuthCode property" on page 654, "DownloadedFile property" on page 653"StreamErrorCode property" on page 657, and "StreamErrorSystem property" on page 657.

Errors may result in no data being sent to the listener.

See also

- "ULFileTransfer class" on page 649
- "ULFileTransfer members" on page 649
- "DownloadFile methods" on page 660
- "DownloadFile() method" on page 660
- "ULFileTransferProgressListener interface" on page 665

ULFileTransferProgressData class

UL Ext.: Returns file transfer progress monitoring data.

Prototypes

Visual Basic
Public Class **ULFileTransferProgressData**

C#
public class **ULFileTransferProgressData**

See also

- "ULFileTransferProgressData members" on page 662
- "ULFileTransferProgressListener interface" on page 665

ULFileTransferProgressData members

Public fields

Member name	Description
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.

Public properties

Member name	Description
BytesReceived property	Returns the number of bytes received so far.
FileSize property	Returns the size of the file being transferred.
Flags property	Returns the current file transfer flags indicating additional information relating to the current state.
ResumedAtSize property	Returns the point in the file where the transfer was resumed.

See also

- "ULFileTransferProgressData class" on page 662
- "ULFileTransferProgressListener interface" on page 665

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.

Prototypes

Visual Basic
Public Shared **FLAG_IS_BLOCKING** As Integer

C#
public const int **FLAG_IS_BLOCKING** ;

See also

- "ULFileTransferProgressData class" on page 662
- "ULFileTransferProgressData members" on page 662

BytesReceived property

Returns the number of bytes received so far.

Prototypes

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 662
- "ULFileTransferProgressData members" on page 662

FileSize property

Returns the size of the file being transferred.

Prototypes

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 662
- "ULFileTransferProgressData members" on page 662

Flags property

Returns the current file transfer flags indicating additional information relating to the current state.

Prototypes

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 662
- "ULFileTransferProgressData members" on page 662
- "FLAG_IS_BLOCKING field" on page 662

ResumedAtSize property

Returns the point in the file where the transfer was resumed.

Prototypes

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 662
- "ULFileTransferProgressData members" on page 662

ULFileTransferProgressListener interface

UL Ext.: The listener interface for receiving file transfer progress events.

Prototypes

Visual Basic
Public Interface **ULFileTransferProgressListener**

C#
public interface **ULFileTransferProgressListener**

See also

- "ULFileTransferProgressListener members" on page 665
- "DownloadFile(ULFileTransferProgressListener) method" on page 660

ULFileTransferProgressListener members

Public methods

Member name	Description
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.

See also

- "ULFileTransferProgressListener interface" on page 665
- "DownloadFile(ULFileTransferProgressListener) method" on page 660

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.

Prototypes

Visual Basic
```
Public Function FileTransferProgressed( _
  ByVal data As ULFileTransferProgressData _
) As Boolean
```

C#
```
public bool FileTransferProgressed(
  ULFileTransferProgressData data
);
```


Parameters

- **data** A "ULFileTransferProgressData class" on page 662 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 665
- "ULFileTransferProgressListener members" on page 665

ULIndexSchema class

UL Ext.: Represents the schema of an UltraLite table index. This class cannot be inherited.

Prototypes

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 "PrimaryKey property" on page 852, "GetIndex method" on page 854, and "GetOptimalIndex method" on page 855 of the "ULTableSchema class" on page 849.

See also

- "ULIndexSchema members" on page 667

ULIndexSchema members

Public properties

Member name	Description
ColumnCount property	Returns the number of columns in the index.
IsForeignKey property	Checks whether the index is a foreign key.
IsForeignKeyCheckOnCommit property	Checks whether referential integrity for the foreign key is performed on commits or on inserts and updates.
IsForeignKeyNullable property	Checks whether the foreign key is nullable.
IsOpen property	Determines whether the index schema is open or closed.
IsPrimaryKey property	Checks whether the index is the primary key.
IsUniqueIndex property	Checks whether the index is unique.
IsUniqueKey property	Checks whether the index is a unique key.
Name property	Returns the name of the index.
ReferencedIndexName property	The name of the referenced primary index if the index is a foreign key.
ReferencedTableName property	The name of the referenced primary table if the index is a foreign key.

Public methods

Member name	Description
GetColumnName method	Returns the name of the *colOrdinalInIndex*'th column in this index.
IsColumnDescending method	Checks whether the named column is used in descending order by the index.

See also

- "ULIndexSchema class" on page 667

ColumnCount property

Returns the number of columns in the index.

Prototypes

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 667
- "ULIndexSchema members" on page 667

IsForeignKey property

Checks whether the index is a foreign key.

Prototypes

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 667
- "ULIndexSchema members" on page 667

IsForeignKeyCheckOnCommit property

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

Prototypes

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 667
- "ULIndexSchema members" on page 667
- "IsForeignKey property" on page 668

IsForeignKeyNullable property

Checks whether the foreign key is nullable.

Prototypes

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 667
- "ULIndexSchema members" on page 667
- "IsForeignKey property" on page 668

IsOpen property

Determines whether the index schema is open or closed.

Prototypes

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 667
- "ULIndexSchema members" on page 667

IsPrimaryKey property

Checks whether the index is the primary key.

Prototypes

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 667
- "ULIndexSchema members" on page 667

IsUniqueIndex property

Checks whether the index is unique.

Prototypes

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 667
- "ULIndexSchema members" on page 667

IsUniqueKey property

Checks whether the index is a unique key.

Prototypes

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 667
- "ULIndexSchema members" on page 667

Name property

Returns the name of the index.

Prototypes

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 667
- "ULIndexSchema members" on page 667

ReferencedIndexName property

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

Prototypes

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 667
- "ULIndexSchema members" on page 667
- "IsForeignKey property" on page 668

ReferencedTableName property

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

Prototypes

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 667
- "ULIndexSchema members" on page 667
- "IsForeignKey property" on page 668

GetColumnName method

Returns the name of the

colOrdinalInIndex'th column in this index.

Prototypes

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 property" on page 668].

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 667
- "ULIndexSchema members" on page 667
- "ColumnCount property" on page 668

IsColumnDescending method

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

Prototypes

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 667
- "ULIndexSchema members" on page 667
- "GetColumnName method" on page 673
- "ColumnCount property" on page 668

ULInfoMessageEventArgs class

Provides data for the "InfoMessage event" on page 528. This class cannot be inherited.

Prototypes

Visual Basic
Public NotInheritable Class **ULInfoMessageEventArgs**
  Inherits EventArgs

C#
public sealed class **ULInfoMessageEventArgs** : EventArgs

See also

- "ULInfoMessageEventArgs members" on page 675

ULInfoMessageEventArgs members

Public properties

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

Public methods

Member name	Description
ToString method	Returns the string representation of the "InfoMessage event" on page 528.

See also

- "ULInfoMessageEventArgs class" on page 675

Message property

The informational or warning message string returned by the database.

Prototypes

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 675
- "ULInfoMessageEventArgs members" on page 675

NativeError property

The SQL code corresponding to the informational message or warning returned by the database.

Prototypes

Visual Basic
Public Readonly Property **NativeError** As ULSQLCode

C#
public ULSQLCode **NativeError** { get;}

Property value

An informational or warning "ULSQLCode enumeration" on page 763 value.

See also

- "ULInfoMessageEventArgs class" on page 675
- "ULInfoMessageEventArgs members" on page 675

Source property

The name of the ADO.NET data provider returning the message.

Prototypes

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 675
- "ULInfoMessageEventArgs members" on page 675

ToString method

Returns the string representation of the "InfoMessage event" on page 528.

Prototypes

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 675
- "ULInfoMessageEventArgs members" on page 675

ULInfoMessageEventHandler delegate

Represents the method that will handle the "InfoMessage event" on page 528.

Prototypes

Visual Basic

```
Public Delegate Sub ULInfoMessageEventHandler( _
   ByVal obj As Object, _
   ByVal args As ULInfoMessageEventArgs _
)
```

C#

```
public delegate void ULInfoMessageEventHandler(
   object obj,
   ULInfoMessageEventArgs args
);
```

ULMetaDataCollectionNames class

Provides a list of constants for use with the "GetSchema(String, String[]) method" on page 522 to retrieve metadata collections. This class cannot be inherited.

Prototypes

Visual Basic
Public NotInheritable Class **ULMetaDataCollectionNames**

C#
public sealed class **ULMetaDataCollectionNames**

See also

- "ULMetaDataCollectionNames members" on page 679

ULMetaDataCollectionNames members

Public properties

Member name	Description
Columns property	Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the Columns collection.
DataSourceInformation property	Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the DataSourceInformation collection.
DataTypes property	Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the DataTypes collection.
ForeignKeys property	Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the ForeignKeys collection.
IndexColumns property	Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the IndexColumns collection.
Indexes property	Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the Indexes collection.
MetaDataCollections property	Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the MetaDataCollections collection.
Publications property	Provides a list of constants for use with the "GetSchema(String, String []) method" on page 522 to retrieve metadata collections.
ReservedWords property	Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the ReservedWords collection.

Member name	Description
Restrictions property	Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the Restrictions collection.
Tables property	Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the Tables collection.

See also

- "ULMetaDataCollectionNames class" on page 679

Columns property

Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the Columns collection.

Prototypes

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 679
- "ULMetaDataCollectionNames members" on page 679

DataSourceInformation property

Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the DataSourceInformation collection.

Prototypes

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 679
- "ULMetaDataCollectionNames members" on page 679

DataTypes property

Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the DataTypes collection.

Prototypes

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 679
- "ULMetaDataCollectionNames members" on page 679

ForeignKeys property

Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the ForeignKeys collection.

Prototypes

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 679
- "ULMetaDataCollectionNames members" on page 679

IndexColumns property

Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the IndexColumns collection.

Prototypes

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 679
- "ULMetaDataCollectionNames members" on page 679

Indexes property

Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the Indexes collection.

Prototypes

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 679
- "ULMetaDataCollectionNames members" on page 679

MetaDataCollections property

Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the MetaDataCollections collection.

Prototypes

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 679
- "ULMetaDataCollectionNames members" on page 679

Publications property

Provides a list of constants for use with the "GetSchema(String, String[]) method" on page 522 to retrieve metadata collections.

Prototypes

Visual Basic
Public Shared Readonly Property **Publications** As String

C#
public const string **Publications** { get;}

See also

- "ULMetaDataCollectionNames class" on page 679
- "ULMetaDataCollectionNames members" on page 679

ReservedWords property

Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the ReservedWords collection.

Prototypes

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 679
- "ULMetaDataCollectionNames members" on page 679

Restrictions property

Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the Restrictions collection.

Prototypes

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 679
- "ULMetaDataCollectionNames members" on page 679

Tables property

Provides a constant for use with the "GetSchema(String) method" on page 521 that represents the Tables collection.

Prototypes

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 679
- "ULMetaDataCollectionNames members" on page 679

ULParameter class

Represents a parameter to a "ULCommand class" on page 462. This class cannot be inherited.

Prototypes

Visual Basic
Public NotInheritable Class **ULParameter**
  Inherits DbParameter

C#
public sealed class **ULParameter** : DbParameter

Remarks

A ULParameter object can be created directly using one of its many constructors, or using the "CreateParameter method" on page 477. 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" on page 689. 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 "CreateParameter method" on page 477) must be added to a "Parameters property" on page 471 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 "Value property" on page 699 is important.

Inherits: DbParameter

Implements: IDbDataParameter, IDataParameter

See also

- "ULParameter members" on page 687

ULParameter members

Public constructors

Member name	Description
ULParameter constructors	Initializes a new instance of the "ULParameter class" on page 687.

Public properties

Member name	Description
DbType property	Specifies the DbType of the parameter
Direction property	A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.
IsNullable property	Specifies whether the parameter accepts null values.
Offset property	Specifies the offset to the "Value property" on page 699.
ParameterName property	Specifies the name of the parameter.
Precision property	Specifies the maximum number of digits used to represent the "Value property" on page 699.
Scale property	Specifies the number of decimal places to which "Value property" on page 699 is resolved.
Size property	Specifies the maximum size of the data within the column.
SourceColumn property	Specifies the name of the source column mapped to the DataSet and used for loading or returning the value.
SourceColumnNullMapping property	
SourceVersion property	The DataRowVersion to use when loading "Value property" on page 699.
ULDbType property	Specifies the "ULDbType enumeration" on page 637 of the parameter
Value property	Specifies the value of the parameter.

Public methods

Member name	Description
ResetDbType method	This method is not supported in UltraLite.NET.
ToString method	Returns the string representation of this instance.

See also

- "ULParameter class" on page 687

ULParameter constructors

Initializes a new instance of the "ULParameter class" on page 687.

ULParameter() constructor

Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.

Prototypes

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 class" on page 462 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 687
- "ULParameter members" on page 687
- "ULParameter constructors" on page 688
- "Value property" on page 699
- "ULParameter(String, Object) constructor" on page 689

ULParameter(String, Object) constructor

Initializes a ULParameter object with the specified parameter name and value.

Prototypes

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 class" on page 462.
- **value** An 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 class" on page 462 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 687
- "ULParameter members" on page 687
- "ULParameter constructors" on page 688
- "ULParameter() constructor" on page 689

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.

Prototypes

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 class" on page 462.

- **dbType** One of the "ULDbType enumeration" on page 637 values.

Remarks

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687
- "ULParameter constructors" on page 688
- "ULParameter() constructor" on page 689
- "ULParameter(String, Object) constructor" on page 689

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.

Prototypes

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 class" on page 462.
- **dbType** One of the "ULDbType enumeration" on page 637 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 "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

- "ULParameter constructors" on page 688
- "ULParameter() constructor" on page 689
- "ULParameter(String, Object) constructor" on page 689

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.

Prototypes

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 class" on page 462.
- **dbType** One of the "ULDbType enumeration" on page 637 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 "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687
- "ULParameter constructors" on page 688
- "ULParameter() constructor" on page 689
- "ULParameter(String, Object) constructor" on page 689

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.

Prototypes

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 class" on page 462.
- **dbType** One of the "ULDbType enumeration" on page 637 values.
- **size** The length of the parameter.
- **direction** One of the 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 DataRowVersion values.
- **value** An Object that is to be the value of the parameter.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687
- "ULParameter constructors" on page 688
- "ULParameter() constructor" on page 689
- "ULParameter(String, Object) constructor" on page 689

DbType property

Specifies the DbType of the parameter

Prototypes

Visual Basic
Public Overrides Property **DbType** As DbType

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

Property value

One of the DbType values.

Remarks

The "ULDbType property" on page 699 and DbType properties are linked. Therefore, setting the DbType changes the "ULDbType property" on page 699 to a supporting "ULDbType enumeration" on page 637.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

Direction property

A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

Prototypes

Visual Basic
Public Overrides Property **Direction** As ParameterDirection

C#
public override ParameterDirection **Direction** { get; set; }

Property value

One of the ParameterDirection values.

Remarks

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

IsNullable property

Specifies whether the parameter accepts null values.

Prototypes

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 "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

Offset property

Specifies the offset to the "Value property" on page 699.

Prototypes

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 "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

ParameterName property

Specifies the name of the parameter.

Prototypes

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 class" on page 462. 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 687
- "ULParameter members" on page 687

Precision property

Specifies the maximum number of digits used to represent the "Value property" on page 699.

Prototypes

Visual Basic
Public Property **Precision** As Byte

C#
public byte **Precision** { get; set; }

Property value

The maximum number of digits used to represent the "Value property" on page 699. The default value is 0, which indicates that the data provider sets the precision for the "Value property" on page 699.

Remarks

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

Scale property

Specifies the number of decimal places to which "Value property" on page 699 is resolved.

Prototypes

Visual Basic
Public Property **Scale** As Byte

C#
public byte **Scale** { get; set; }

Property value

The number of decimal places to which "Value property" on page 699 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 "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

Size property

Specifies the maximum size of the data within the column.

Prototypes

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 "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

SourceColumn property

Specifies the name of the source column mapped to the DataSet and used for loading or returning the value.

Prototypes

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 "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

SourceColumnNullMapping property

Prototypes

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

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

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

SourceVersion property

The DataRowVersion to use when loading "Value property" on page 699.

Prototypes

Visual Basic
Public Overrides Property **SourceVersion** As DataRowVersion

C#
public override DataRowVersion **SourceVersion** { get; set; }

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

ULDbType property

Specifies the "ULDbType enumeration" on page 637 of the parameter

Prototypes

Visual Basic
Public Property **ULDbType** As ULDbType

C#
public ULDbType **ULDbType** { get; set; }

Property value

One of the "ULDbType enumeration" on page 637 values.

Remarks

The ULDbType and "DbType property" on page 694 are linked. Therefore, setting the ULDbType changes the "DbType property" on page 694 to a supporting DbType.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

Value property

Specifies the value of the parameter.

Prototypes

Visual Basic
Public Overrides Property **Value** As Object

C#
public override object **Value** { get; set; }

Property value

An 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 class" on page 640 with "ULSQLCode enumeration" on page 763 if it cannot convert the value.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

ResetDbType method

This method is not supported in UltraLite.NET.

Prototypes

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 "Value property" on page 699 is important.

See also

- "ULParameter class" on page 687
- "ULParameter members" on page 687

ToString method

Returns the string representation of this instance.

Prototypes

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 687
- “ULParameter members” on page 687

ULParameterCollection class

Represents all parameters to a "ULCommand class" on page 462. This class cannot be inherited.

Prototypes

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 "CommandText property" on page 467. 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 "CommandText property" on page 467 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 "Parameters property" on page 471.

Inherits: DbParameterCollection

Implements: IDataParameterCollection

See also

- "ULParameterCollection members" on page 702

ULParameterCollection members

Public properties

Member name	Description
Count property	Returns the number of "ULParameter class" on page 687 objects in the collection.
IsFixedSize property	Indicates whether the ULParameterCollection has a fixed size.
IsReadOnly property	Indicates whether the ULParameterCollection is read-only.
IsSynchronized property	Indicates whether the ULParameterCollection is synchronized.
Item properties	Gets and sets a DbParameter in the collection.
SyncRoot property	Returns an object that can be used to synchronize access to the SAParameterCollection.

Public methods

Member name	Description
Add methods	Adds a "ULParameter class" on page 687 to the collection.
AddRange methods	Adds an array of values to the end of the ULParameterCollection.
Clear method	Removes all the parameters from the collection.
Contains methods	Indicates whether a DbParameter with the specified property exists in the collection.
CopyTo method	Copies "ULParameter class" on page 687 objects from the ULParameterCollection to the specified array.
GetEnumerator method	Returns an enumerator for the collection.
IndexOf methods	Returns the index of the specified DbParameter object.
Insert method	Inserts an "ULParameter class" on page 687 in the collection at the specified index.
Remove method	Removes an "ULParameter class" on page 687 from the collection.
RemoveAt methods	Removes a specified DbParameter object from the collection.

See also

- "ULParameterCollection class" on page 702

Count property

Returns the number of "ULParameter class" on page 687 objects in the collection.

Prototypes

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

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

Property value

The number of "ULParameter class" on page 687 objects in the collection.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702

IsFixedSize property

Indicates whether the ULParameterCollection has a fixed size.

Prototypes

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 702
- "ULParameterCollection members" on page 702

IsReadOnly property

Indicates whether the ULParameterCollection is read-only.

Prototypes

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 702
- "ULParameterCollection members" on page 702

IsSynchronized property

Indicates whether the ULParameterCollection is synchronized.

Prototypes

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 702
- "ULParameterCollection members" on page 702

Item properties

Gets and sets a DbParameter in the collection.

Item(Int32) property

Returns the "ULParameter class" on page 687 at the specified index. In C#, this property is the indexer for the ULParameterCollection class.

Prototypes

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,"Count property" on page 703-1]. The first parameter in the collection has an index value of zero.

Property value

The "ULParameter class" on page 687 at the specified index.

Remarks

This is the strongly-typed version of DbParameterCollection.Item.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "Item properties" on page 705

Item(String) property

Returns the "ULParameter class" on page 687 with the specified name. In C#, this property is the indexer for the ULParameterCollection class.

Prototypes

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 class" on page 687 with the specified name.

Remarks

This is the strongly-typed version of DbParameterCollection.Item.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "Item properties" on page 705
- "Item(Int32) property" on page 608
- "GetOrdinal method" on page 624
- "GetValue method" on page 629
- "GetFieldType method" on page 619

SyncRoot property

Returns an object that can be used to synchronize access to the SAParameterCollection.

Prototypes

Visual Basic

```
Public Overrides Readonly Property SyncRoot As Object
```

C#

```
public override object SyncRoot  { get;}
```

Property value

The object to be used to synchornize access to this collection.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702

Add methods

Adds a "ULParameter class" on page 687 to the collection.

Add(Object) method

Adds a "ULParameter class" on page 687 to the collection.

Prototypes

Visual Basic

```
Public Overrides Function Add( _
  ByVal value As Object _
) As Integer
```

C#

```
public override int Add(
  object value
);
```

Parameters

- **value** The "ULParameter class" on page 687 object to add to the collection.

Return value

The index of the new "ULParameter class" on page 687 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 "CommandText property" on page 467. 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 "CommandText property" on page 467 as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "Add methods" on page 707
- "Add(ULParameter) method" on page 708
- "Add(String, Object) method" on page 708

Add(ULParameter) method

Adds a "ULParameter class" on page 687 to the collection.

Prototypes

Visual Basic
```
Public Function Add( _
  ByVal value As ULParameter _
) As ULParameter
```

C#
```
public ULParameter Add(
  ULParameter value
);
```

Parameters

- **value** The "ULParameter class" on page 687 object to add to the collection.

Return value

The new "ULParameter class" on page 687 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 "CommandText property" on page 467. 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 "CommandText property" on page 467 as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "Add methods" on page 707
- "Add(String, Object) method" on page 708

Add(String, Object) method

Adds a new "ULParameter class" on page 687, created using the specified parameter name and value, to the collection.

Prototypes

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 class" on page 462.
- **value** An Object that is to be the value of the parameter.

Return value

The new "ULParameter class" on page 687 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 "CommandText property" on page 467. 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 "CommandText property" on page 467 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 class" on page 462 called cmd.

```
' Visual Basic
cmd.Parameters.Add( "", CType( 0, Object ) )

// C#
cmd.Parameters.Add( "", (object)0 );
```

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "Add methods" on page 707
- "Add(ULParameter) method" on page 708

Add(String, ULDbType) method

Adds a new "ULParameter class" on page 687, created using the specified parameter name and data type, to the collection.

Prototypes

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 class" on page 462.
- **ulDbType** One of the "ULDbType enumeration" on page 637 values.

Return value

The new "ULParameter class" on page 687 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 "CommandText property" on page 467. 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 "CommandText property" on page 467 as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "Add methods" on page 707
- "Add(ULParameter) method" on page 708
- "Add(String, Object) method" on page 708

Add(String, ULDbType, Int32) method

Adds a new "ULParameter class" on page 687, created using the specified parameter name, data type, and length, to the collection.

Prototypes

```
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 class" on page 462.
- **ulDbType** One of the "ULDbType enumeration" on page 637 values.
- **size** The length of the parameter.

Return value

The new "ULParameter class" on page 687 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 "CommandText property" on page 467. 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 "CommandText property" on page 467 as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "Add methods" on page 707
- "Add(ULParameter) method" on page 708
- "Add(String, Object) method" on page 708

Add(String, ULDbType, Int32, String) method

Adds a new "ULParameter class" on page 687, created using the specified parameter name, data type, length, and source column name, to the collection.

Prototypes

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 class" on page 462.
- **ulDbType** One of the "ULDbType enumeration" on page 637 values.
- **size** The length of the parameter.
- **sourceColumn** The name of the source column to map.

Return value

The new "ULParameter class" on page 687 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 "CommandText property" on page 467. 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 "CommandText property" on page 467 as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "Add methods" on page 707
- "Add(ULParameter) method" on page 708
- "Add(String, Object) method" on page 708

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.

Prototypes

Visual Basic

```
Public Overrides Sub AddRange( _
  ByVal values As Array _
)
```


C#
public override void **AddRange(**
  Array *values*
);

Parameters

- **values** An array of "ULParameter class" on page 687 objects to add to the end of this collection.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "AddRange methods" on page 712

AddRange(ULParameter[]) method

Adds an array of values to the end of the ULParameterCollection.

Prototypes

Visual Basic
Public Sub **AddRange(** _
  ByVal *values* As ULParameter() _
)

C#
public void **AddRange(**
  ULParameter[] *values*
);

Parameters

- **values** An array of "ULParameter class" on page 687 objects to add to the end of this collection.

Remarks

This is the strongly-typed version of DbParameterCollection.AddRange.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "AddRange methods" on page 712

Clear method

Removes all the parameters from the collection.

Prototypes

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

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

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702

Contains methods

Indicates whether a DbParameter with the specified property exists in the collection.

Contains(Object) method

Checks whether a "ULParameter class" on page 687 exists in the collection.

Prototypes

Visual Basic
Public Overrides Function **Contains(** _
  ByVal *value* As Object _
) As Boolean

C#
public override bool **Contains(**
  object *value*
);

Parameters

- **value** The "ULParameter class" on page 687 object to check for.

Return value

True if the collection contains the "ULParameter class" on page 687, false otherwise.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "Contains methods" on page 714
- "Contains(String) method" on page 714

Contains(String) method

Checks whether a "ULParameter class" on page 687 with the specified name exists in the collection.

Prototypes

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 class" on page 687, false otherwise.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "Contains methods" on page 714
- "Contains(Object) method" on page 714

CopyTo method

Copies "ULParameter class" on page 687 objects from the ULParameterCollection to the specified array.

Prototypes

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 class" on page 687 objects.
- **index** The starting index of the array.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702

GetEnumerator method

Returns an enumerator for the collection.

Prototypes

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 702
- "ULParameterCollection members" on page 702

IndexOf methods

Returns the index of the specified DbParameter object.

IndexOf(Object) method

Returns the location of the "ULParameter class" on page 687 in the collection.

Prototypes

Visual Basic
```
Public Overrides Function IndexOf( _
  ByVal value As Object _
) As Integer
```

C#
```
public override int IndexOf(
  object value
);
```

Parameters

- **value** The "ULParameter class" on page 687 object to locate.

Return value

The zero-based index of the "ULParameter class" on page 687 in the collection or -1 if the parameter is not found.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "IndexOf methods" on page 716
- "IndexOf(String) method" on page 717

IndexOf(String) method

Returns the location of the "ULParameter class" on page 687 with the specified name in the collection.

Prototypes

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 class" on page 687 in the collection or -1 if the parameter is not found.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "IndexOf methods" on page 716
- "IndexOf(Object) method" on page 716

Insert method

Inserts an "ULParameter class" on page 687 in the collection at the specified index.

Prototypes

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 class" on page 687 object to insert.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702

Remove method

Removes an "ULParameter class" on page 687 from the collection.

Prototypes

Visual Basic

```
Public Overrides Sub Remove( _
  ByVal value As Object _
)
```

C#

```
public override void Remove(
  object value
);
```

Parameters

- **value** The "ULParameter class" on page 687 object to remove.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702

RemoveAt methods

Removes a specified DbParameter object from the collection.

RemoveAt(Int32) method

Removes the parameter at the specified index in the collection.

Prototypes

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,"Count property" on page 703-1]. The first parameter in the collection has an index value of zero.

See also

- "ULParameterCollection class" on page 702
- "ULParameterCollection members" on page 702
- "RemoveAt methods" on page 718
- "RemoveAt(String) method" on page 719

RemoveAt(String) method

Removes the parameter with the specified name from the collection.

Prototypes

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 702
- "ULParameterCollection members" on page 702
- "RemoveAt methods" on page 718
- "RemoveAt(Int32) method" on page 718

ULPublicationSchema class

UL Ext.: Represents the schema of an UltraLite publication. This class cannot be inherited.

Prototypes

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 "GetPublicationSchema method" on page 598 of the "ULDatabaseSchema class" on page 592.

UltraLite methods requiring a publication mask actually require a set of publications to check. A set is formed by or'ing the publication masks of individual publications. For example:

```
' Visual Basic
Dim mask As Integer = pub1.Mask Or pub2.Mask

// C#
int mask = pub1.Mask | pub2.Mask;
```

Two special mask values are also provided by this class. "SYNC_ALL_DB field" on page 721 corresponds to the entire database. "SYNC_ALL_PUBS field" on page 721 corresponds to all publications.

Note: Publication masks may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached masks after a schema upgrade.

See also

- "ULPublicationSchema members" on page 720
- "Mask property" on page 722
- "GetLastDownloadTime method" on page 519
- "CountUploadRows(Int32, UInt32) method" on page 513
- "Schema property" on page 507

ULPublicationSchema members

Public fields

Member name	Description
SYNC_ALL_DB field	Publication mask corresponding to the entire database. This field is constant and read-only.
SYNC_ALL_PUBS field	Publication mask corresponding to all publications. This field is constant and read-only.

Public properties

Member name	Description
IsOpen property	Determines whether the publication schema is open or closed.
Mask property	Returns the publication mask of the publication.
Name property	Returns the name of this publication.

See also

- "ULPublicationSchema class" on page 720
- "Mask property" on page 722
- "GetLastDownloadTime method" on page 519
- "CountUploadRows(Int32, UInt32) method" on page 513
- "Schema property" on page 507

SYNC_ALL_DB field

Publication mask corresponding to the entire database. This field is constant and read-only.

Prototypes

Visual Basic
Public Shared **SYNC_ALL_DB** As Integer

C#
public const int **SYNC_ALL_DB** ;

See also

- "ULPublicationSchema class" on page 720
- "ULPublicationSchema members" on page 720

SYNC_ALL_PUBS field

Publication mask corresponding to all publications. This field is constant and read-only.

Prototypes

Visual Basic
Public Shared **SYNC_ALL_PUBS** As Integer

C#
public const int **SYNC_ALL_PUBS** ;

See also

- "ULPublicationSchema class" on page 720
- "ULPublicationSchema members" on page 720

IsOpen property

Determines whether the publication schema is open or closed.

Prototypes

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 720
- "ULPublicationSchema members" on page 720

Mask property

Returns the publication mask of the publication.

Prototypes

Visual Basic
Public Readonly Property **Mask** As Integer

C#
public int **Mask** { get;}

Property value

The publication mask of the publication.

Remarks

Publication IDs, masks, and counts may change during a schema upgrade. To correctly identify a publication, access it by name, or refresh the cached masks and counts after a schema upgrade.

See also

- "ULPublicationSchema class" on page 720
- "ULPublicationSchema members" on page 720

Name property

Returns the name of this publication.

Prototypes

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 720
- "ULPublicationSchema members" on page 720

ULResultSet class

UL Ext.: Represents an editable result set in an UltraLite database.

Prototypes

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 "ExecuteResultSet() method" on page 486 of the "ULCommand class" on page 462.

```
' 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 class" on page 724 object represents an editable result set on which you can perform positioned updates and deletes. For fully editable result sets, use "ExecuteTable() method" on page 489 or a "ULDataAdapter class" on page 574.

Inherits: "ULDataReader class" on page 602

Implements: IDataReader, IDataRecord, IDisposable

See also

- "ULResultSet members" on page 724

ULResultSet members

Public properties

Member name	Description
Depth property (inherited from ULDataReader)	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
FieldCount property (inherited from ULDataReader)	Returns the number of columns in the cursor.

Member name	Description
HasRows property (inherited from ULDataReader)	Checks whether the ULDataReader has one or more rows.
IsBOF property (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is before the first row.
IsClosed property (inherited from ULDataReader)	Checks whether the cursor is currently open.
IsEOF property (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is after the last row.
Item properties (inherited from ULDataReader)	Gets the value of a specified column as an instance of Object.
RecordsAffected property (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 (inherited from ULDataReader)	**UL Ext.:** Returns the number of rows in the cursor.
Schema property (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	Appends the specified subset of the specified array of Bytes to the new value for the specified “ULDbType enumeration” on page 637 column.
AppendChars method	Appends the specified subset of the specified array of Chars to the new value for the specified “ULDbType enumeration” on page 637 column.
Close method (inherited from ULDataReader)	Closes the cursor.
Delete method	Deletes the current row.
Dispose (inherited from DbDataReader)	Releases the resources consumed by this DbDataReader.
GetBoolean method (inherited from ULDataReader)	Returns the value for the specified column as a Boolean.
GetByte method (inherited from ULDataReader)	Returns the value for the specified column as an unsigned 8-bit value (Byte).

Member name	Description
GetBytes methods (inherited from ULDataReader)	**UL Ext.:** Returns the value for the specified column as an array of Bytes. Only valid for columns of type "ULDbType enumeration" on page 637, "ULDbType enumeration" on page 637, or "ULDbType enumeration" on page 637.
GetChar method (inherited from ULDataReader)	This method is not supported in UltraLite.NET.
GetChars method (inherited from ULDataReader)	Copies a subset of the value for the specified "ULDbType enumeration" on page 637 column, beginning at the specified offset, to the specified offset of the destination Char array.
GetData (inherited from DbDataReader)	Returns a DbDataReader object for the requested column ordinal.
GetDataTypeName method (inherited from ULDataReader)	Returns the name of the specified column's provider data type.
GetDateTime method (inherited from ULDataReader)	Returns the value for the specified column as a DateTime with millisecond accuracy.
GetDecimal method (inherited from ULDataReader)	Returns the value for the specified column as a Decimal.
GetDouble method (inherited from ULDataReader)	Returns the value for the specified column as a Double.
GetEnumerator method (inherited from ULDataReader)	Returns an IEnumerator that iterates through the ULDataReader.
GetFieldType method (inherited from ULDataReader)	Returns the Type most appropriate for the specified column.
GetFloat method (inherited from ULDataReader)	Returns the value for the specified column as a Single.
GetGuid method (inherited from ULDataReader)	Returns the value for the specified column as a UUID (Guid).
GetInt16 method (inherited from ULDataReader)	Returns the value for the specified column as an Int16.
GetInt32 method (inherited from ULDataReader)	Returns the value for the specified column as an Int32.
GetInt64 method (inherited from ULDataReader)	Returns the value for the specified column as an Int64.
GetName method (inherited from ULDataReader)	Returns the name of the specified column.
GetOrdinal method (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 (inherited from ULDataReader)	Returns a DataTable that describes the column metadata of the ULDataReader.
GetString method (inherited from ULDataReader)	Returns the value for the specified column as a String.
GetTimeSpan method (inherited from ULDataReader)	Returns the value for the specified column as a TimeSpan with millisecond accuracy.
GetUInt16 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt16.
GetUInt32 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt32.
GetUInt64 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt64.
GetValue method (inherited from ULDataReader)	Returns the value of the specified column in its native format.
GetValues method (inherited from ULDataReader)	Returns all the column values for the current row.
IsDBNull method (inherited from ULDataReader)	Checks whether the value from the specified column is NULL.
MoveAfterLast method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to after the last row of the cursor.
MoveBeforeFirst method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to before the first row of the cursor.
MoveFirst method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the first row of the cursor.
MoveLast method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the last row of the cursor.
MoveNext method (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 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the previous row or before the first row.

Member name	Description
MoveRelative method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor relative to the current row.
NextResult method (inherited from ULDataReader)	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
Read method (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	Sets the value for the specified column using a Boolean.
SetByte method	Sets the value for the specified column using a Byte (unsigned 8-bit integer).
SetBytes method	Sets the value for the specified column using an array of Bytes.
SetDBNull method	Sets a column to NULL.
SetDateTime method	Sets the value for the specified column using a DateTime.
SetDecimal method	Sets the value for the specified column using a Decimal.
SetDouble method	Sets the value for the specified column using a Double.
SetFloat method	Sets the value for the specified column using a Single.
SetGuid method	Sets the value for the specified column using a Guid.
SetInt16 method	Sets the value for the specified column using an Int16.
SetInt32 method	Sets the value for the specified column using an Int32.
SetInt64 method	Sets the value for the specified column using an Int64.
SetString method	Sets the value for the specified column using a String.
SetTimeSpan method	Sets the value for the specified column using a TimeSpan.
SetToDefault method	Sets the value for the specified column to its default value.
SetUInt16 method	Sets the value for the specified column using a UInt16.
SetUInt32 method	Sets the value for the specified column using an UInt32.
SetUInt64 method	Sets the value for the specified column using a UInt64.
Update method	Updates the current row with the current column values (specified using the set methods).

See also

- "ULResultSet class" on page 724

AppendBytes method

Appends the specified subset of the specified array of Bytes to the new value for the specified "ULDbType enumeration" on page 637 column.

Prototypes

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,"FieldCount property" on page 606-1]. 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, "InsertBegin method" on page 844 initializes the new value to the column's default value. The data in the row is not actually changed until you execute an "Insert method" on page 843, 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 class" on page 640 with code "ULSQLCode enumeration" on page 763 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 class" on page 640 with the appropriate error code is thrown.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

AppendChars method

Appends the specified subset of the specified array of Chars to the new value for the specified "ULDbType enumeration" on page 637 column.

Prototypes

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,"FieldCount property" on page 606-1]. 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, "InsertBegin method" on page 844 initializes the new value to the column's default value. The data in the row is not actually changed until you execute an "Insert method" on page 843, 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 class" on page 640 with code "ULSQLCode enumeration" on page 763 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 class" on page 640 with the appropriate error code is thrown.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

Delete method

Deletes the current row.

Prototypes

Visual Basic

```
Public Sub Delete()
```

C#

```
public void Delete();
```

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "StartSynchronizationDelete method" on page 526
- "StopSynchronizationDelete method" on page 526

SetBoolean method

Sets the value for the specified column using a Boolean.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetByte method

Sets the value for the specified column using a Byte (unsigned 8-bit integer).

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624

- "Schema property" on page 610
- "GetFieldType method" on page 619

SetBytes method

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

Prototypes

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,"FieldCount property" on page 606-1]. 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 enumeration" on page 637 or "ULDbType enumeration" on page 637, or for columns of type "ULDbType enumeration" on page 637 when val is of length 16. The data in the row is not actually changed until you execute an "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetDBNull method

Sets a column to NULL.

Prototypes

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,"FieldCount property" on page 606-1]. The first column in the cursor has an ID value of zero.

Remarks

The data is not actually changed until you execute an "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "IsColumnNullable method" on page 860

SetDateTime method

Sets the value for the specified column using a DateTime.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetDecimal method

Sets the value for the specified column using a Decimal.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetDouble method

Sets the value for the specified column using a Double.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetFloat method

Sets the value for the specified column using a Single.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetGuid method

Sets the value for the specified column using a Guid.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed. Only valid for columns of type "ULDbType enumeration" on page 637 or for columns of type "ULDbType enumeration" on page 637 with length 16.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetNewUUID method" on page 520
- "GetOrdinal method" on page 624
- "Schema property" on page 610

- "GetFieldType method" on page 619
- "GetColumnSize method" on page 572

SetInt16 method

Sets the value for the specified column using an Int16.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetInt32 method

Sets the value for the specified column using an Int32.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetInt64 method

Sets the value for the specified column using an Int64.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetString method

Sets the value for the specified column using a String.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetTimeSpan method

Sets the value for the specified column using a TimeSpan.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetToDefault method

Sets the value for the specified column to its default value.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619
- "GetColumnDefaultValue method" on page 853

SetUInt16 method

Sets the value for the specified column using a UInt16.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "GetFieldType method" on page 619

SetUInt32 method

Sets the value for the specified column using an UInt32.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

SetUInt64 method

Sets the value for the specified column using a UInt64.

Prototypes

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,"FieldCount property" on page 606-1]. 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 "Insert method" on page 843 or "Update method" on page 744, and that change is not made permanent until it is committed.

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724
- "GetOrdinal method" on page 624
- "Schema property" on page 610
- "GetFieldType method" on page 619

Update method

Updates the current row with the current column values (specified using the set methods).

Prototypes

Visual Basic
Public Sub **Update()**

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

See also

- "ULResultSet class" on page 724
- "ULResultSet members" on page 724

ULResultSetSchema class

UL Ext.: Represents the schema of an UltraLite result set. This class cannot be inherited.

Prototypes

Visual Basic
Public NotInheritable Class **ULResultSetSchema**
  Inherits ULCursorSchema

C#
public sealed class **ULResultSetSchema** : ULCursorSchema

Remarks

There is no constructor for this class. A "ULResultSetSchema class" on page 745 object is attached to a result set as its "Schema property" on page 610.

A result set schema is only valid while the data reader is open.

Inherits: "ULCursorSchema class" on page 566

See also

- "ULResultSetSchema members" on page 745
- "ULCommand class" on page 462
- "ULDataReader class" on page 602

ULResultSetSchema members

Public properties

Member name	Description
ColumnCount property (inherited from ULCursorSchema)	Returns the number of columns in the cursor.
IsOpen property (inherited from ULCursorSchema)	Checks whether the cursor schema is currently open.
Name property	Returns the name of the cursor.

Public methods

Member name	Description
GetColumnID method (inherited from ULCursorSchema)	Returns the column ID of the named column.
GetColumnName method (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.

Member name	Description
GetColumnPrecision method (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 (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
GetColumnScale method (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 (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 (inherited from ULCursorSchema)	Returns the UltraLite.NET data type of the column identified by the specified column ID.
GetSchemaTable method (inherited from ULCursorSchema)	Returns a DataTable that describes the column schema of the "ULDataReader class" on page 602.

See also

- "ULResultSetSchema class" on page 745
- "ULCommand class" on page 462
- "ULDataReader class" on page 602

Name property

Returns the name of the cursor.

Prototypes

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 745
- "ULResultSetSchema members" on page 745

ULRowsCopiedEventArgs class

Represents the set of arguments passed to the "ULRowsCopiedEventHandler delegate" on page 750. This class cannot be inherited.

Prototypes

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 747

ULRowsCopiedEventArgs members

Public constructors

Member name	Description
ULRowsCopiedEventArgs constructor	Creates a new instance of the ULRowsCopiedEventArgs object.

Public properties

Member name	Description
Abort property	Gets or sets a value that indicates whether the bulk-copy operation should be aborted.
RowsCopied property	Returns the number of rows copied during the current bulk-copy operation.

See also

- "ULRowsCopiedEventArgs class" on page 747

ULRowsCopiedEventArgs constructor

Creates a new instance of the ULRowsCopiedEventArgs object.

Prototypes

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 747
- "ULRowsCopiedEventArgs members" on page 747

Abort property

Gets or sets a value that indicates whether the bulk-copy operation should be aborted.

Prototypes

```
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 747
- "ULRowsCopiedEventArgs members" on page 747

RowsCopied property

Returns the number of rows copied during the current bulk-copy operation.

Prototypes

```
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 747
- "ULRowsCopiedEventArgs members" on page 747

ULRowsCopiedEventHandler delegate

Represents the method that will handle the "ULRowsCopied event" on page 444.

Prototypes

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.

Object

ULRowUpdatedEventArgs class

Provides data for the "RowUpdated event" on page 582. This class cannot be inherited.

Prototypes

Visual Basic

```
Public NotInheritable Class ULRowUpdatedEventArgs
  Inherits RowUpdatedEventArgs
```

C#

```
public sealed class ULRowUpdatedEventArgs : RowUpdatedEventArgs
```

Remarks

Inherits: RowUpdatedEventArgs

See also

- "ULRowUpdatedEventArgs members" on page 751

ULRowUpdatedEventArgs members

Public constructors

Member name	Description
ULRowUpdatedEventArgs constructor	Initializes a new instance of the ULRowUpdatedEventArgs class.

Public properties

Member name	Description
Command property	Returns the "ULCommand class" on page 462 executed when DbDataAdapter.Update is called.
Errors (inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the RowUpdatedEventArgs.Command was executed.
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.
Row (inherited from RowUpdatedEventArgs)	Gets the DataRow sent through an DbDataAdapter.Update.
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.

Member name	Description
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 751

ULRowUpdatedEventArgs constructor

Initializes a new instance of the ULRowUpdatedEventArgs class.

Prototypes

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 DataRow sent through an DbDataAdapter.Update.
- **command** The IDbCommand executed when DbDataAdapter.Update is called.
- **statementType** One of the StatementType values that specifies the type of query executed.
- **tableMapping** The DataTableMapping sent through an DbDataAdapter.Update.

See also

- "ULRowUpdatedEventArgs class" on page 751

- "ULRowUpdatedEventArgs members" on page 751

Command property

Returns the "ULCommand class" on page 462 executed when DbDataAdapter.Update is called.

Prototypes

Visual Basic
Public Readonly Property **Command** As ULCommand

C#
public ULCommand **Command** { get;}

Property value

The "ULCommand class" on page 462 object executed by the update.

Remarks

This is the strongly-typed version of RowUpdatedEventArgs.Command.

See also

- "ULRowUpdatedEventArgs class" on page 751
- "ULRowUpdatedEventArgs members" on page 751

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.

Prototypes

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 751
- "ULRowUpdatedEventArgs members" on page 751

ULRowUpdatedEventHandler delegate

Represents the method that will handle the "RowUpdated event" on page 582.

Prototypes

Visual Basic

```
Public Delegate Sub ULRowUpdatedEventHandler( _
   ByVal sender As Object, _
   ByVal e As ULRowUpdatedEventArgs _
)
```

C#

```
public delegate void ULRowUpdatedEventHandler(
   object sender,
   ULRowUpdatedEventArgs e
);
```

ULRowUpdatingEventArgs class

Provides data for the "RowUpdating event" on page 583. This class cannot be inherited.

Prototypes

Visual Basic
Public NotInheritable Class **ULRowUpdatingEventArgs**
  Inherits RowUpdatingEventArgs

C#
public sealed class **ULRowUpdatingEventArgs** : RowUpdatingEventArgs

Remarks

Inherits: RowUpdatingEventArgs

See also

- "ULRowUpdatingEventArgs members" on page 755

ULRowUpdatingEventArgs members

Public constructors

Member name	Description
ULRowUpdatingEventArgs constructor	Initializes a new instance of the ULRowUpdatingEventArgs class.

Public properties

Member name	Description
Command property	Specifies the "ULCommand class" on page 462 to execute when performing the DbDataAdapter.Update.
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.
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 755

ULRowUpdatingEventArgs constructor

Initializes a new instance of the ULRowUpdatingEventArgs class.

Prototypes

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 DataRow to update.
- **command** The IDbCommand to execute during the update.
- **statementType** One of the StatementType values that specifies the type of query executed.
- **tableMapping** The DataTableMapping sent through an DbDataAdapter.Update.

See also

- "ULRowUpdatingEventArgs class" on page 755
- "ULRowUpdatingEventArgs members" on page 755

Command property

Specifies the "ULCommand class" on page 462 to execute when performing the DbDataAdapter.Update.

Prototypes

Visual Basic

```
Public Property Command As ULCommand
```

C#

```
public ULCommand Command  { get; set; }
```


Property value

The "ULCommand class" on page 462 object to execute when updating.

Remarks

This is the strongly-typed version of RowUpdatingEventArgs.Command.

See also

- "ULRowUpdatingEventArgs class" on page 755
- "ULRowUpdatingEventArgs members" on page 755

ULRowUpdatingEventHandler delegate

Represents the method that will handle the "RowUpdating event" on page 583.

Prototypes

Visual Basic

```
Public Delegate Sub ULRowUpdatingEventHandler( _
   ByVal sender As Object, _
   ByVal e As ULRowUpdatingEventArgs _
)
```

C#

```
public delegate void ULRowUpdatingEventHandler(
   object sender,
   ULRowUpdatingEventArgs e
);
```

ULRuntimeType enumeration

UL Ext.: Enumerates the types of UltraLite.NET runtimes.

Prototypes

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_CLIENT	Selects the UltraLite engine runtime. The UltraLite.NET engine client communicates with the UltraLite engine to access databases. This means that databases can be shared by different applications.	1

See also

- "RuntimeType property" on page 586

ULServerSyncListener interface

UL Ext.: The listener interface for receiving server synchronization messages.

Prototypes

Visual Basic
Public Interface **ULServerSyncListener**

C#
public interface **ULServerSyncListener**

See also

- "ULServerSyncListener members" on page 760

ULServerSyncListener members

Public methods

Member name	Description
ServerSyncInvoked method	Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

See also

- "ULServerSyncListener interface" on page 760

ServerSyncInvoked method

Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

Prototypes

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 760
- "ULServerSyncListener members" on page 760

ULSQLCode enumeration

UL Ext.: Enumerates the SQL codes that may be reported by UltraLite.NET.

Prototypes

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" [*SQL Anywhere 10 - Error Messages*].	-150
SQLE_ALIAS_NOT_UNIQUE	See "Alias '%1' is not unique" [*SQL Anywhere 10 - Error Messages*].	-830
SQLE_ALIAS_NOT_YET_DEFINED	See "Definition for alias '%1' must appear before its first reference" [*SQL Anywhere 10 - Error Messages*].	-831
SQLE_AMBIGUOUS_INDEX_NAME	See "Index name '%1' is ambiguous" [*SQL Anywhere 10 - Error Messages*].	-678
SQLE_ARGUMENT_CANNOT_BE_NULL	See "Argument %1 of procedure '%2' cannot be NULL" [*SQL Anywhere 10 - Error Messages*].	-90
SQLE_BAD_ENCRYPTION_KEY	See "Incorrect or missing encryption key" [*SQL Anywhere 10 - Error Messages*].	-840
SQLE_BAD_PARAM_INDEX	See "Input parameter index out of range" [*SQL Anywhere 10 - Error Messages*].	-689
SQLE_CANNOT_ACCESS_FILESYSTEM	See "Unable to access the filesystem on the device" [*SQL Anywhere 10 - Error Messages*].	-1108
SQLE_CANNOT_CHANGE_ML_REMOTE_ID	See "Cannot change the MobiLink remote id when the status of the last upload is unknown" [*SQL Anywhere 10 - Error Messages*].	-1118
SQLE_CANNOT_CONVERT	See "Invalid data conversion" [*SQL Anywhere 10 - Error Messages*].	103
SQLE_CANNOT_EXECUTE_STMT	See "Statement cannot be executed" [*SQL Anywhere 10 - Error Messages*].	111
SQLE_CANNOT_MODIFY	See "Cannot modify column '%1' in table '%2'" [*SQL Anywhere 10 - Error Messages*].	-191

Member name	Description	Value
SQLE_CANNOT_REGISTER_LISTENER	See “The specified listener could not be registered” [*SQL Anywhere 10 - Error Messages*].	-992
SQLE_CLIENT_OUT_OF_MEMORY	See “Client out of memory” [*SQL Anywhere 10 - Error Messages*].	-876
SQLE_COLUMN_AMBIGUOUS	See “Column '%1' found in more than one table -- need a correlation name” [*SQL Anywhere 10 - Error Messages*].	-144
SQLE_COLUMN_CANNOT_BE_NULL	See “Column '%1' in table '%2' cannot be NULL” [*SQL Anywhere 10 - Error Messages*].	-195
SQLE_COLUMN_IN_INDEX	See “Cannot alter a column in an index” [*SQL Anywhere 10 - Error Messages*].	-127
SQLE_COLUMN_NOT_FOUND	See “Column '%1' not found” [*SQL Anywhere 10 - Error Messages*].	-143
SQLE_COLUMN_NOT_INDEXED	See “Column '%1' not part of any indexes in its containing table” [*SQL Anywhere 10 - Error Messages*].	-1101
SQLE_COLUMN_NOT_STREAMABLE	See “The operation failed because column '%1''s type does not support streaming” [*SQL Anywhere 10 - Error Messages*].	-1100
SQLE_COMMUNICATIONS_ERROR	See “Communication error” [*SQL Anywhere 10 - Error Messages*].	-85
SQLE_CONNECTION_ALREADY_EXISTS	See “This connection already exists” [*SQL Anywhere 10 - Error Messages*].	-955
SQLE_CONNECTION_NOT_FOUND	See “Connection not found” [*SQL Anywhere 10 - Error Messages*].	-108
SQLE_CONNECTION_RESTORED	See “UltraLite connection was restored” [*SQL Anywhere 10 - Error Messages*].	133
SQLE_CONSTRAINT_NOT_FOUND	See “Constraint '%1' not found” [*SQL Anywhere 10 - Error Messages*].	-929
SQLE_CONVERSION_ERROR	See “Cannot convert %1 to a %2” [*SQL Anywhere 10 - Error Messages*].	-157
SQLE_COULD_NOT_FIND_FUNCTION	See “Could not find '%1' in dynamic library '%2'” [*SQL Anywhere 10 - Error Messages*].	-621
SQLE_COULD_NOT_LOAD_LIBRARY	See “Could not load dynamic library '%1'” [*SQL Anywhere 10 - Error Messages*].	-620
SQLE_CURSOR_ALREADY_OPEN	See “Cursor already open” [*SQL Anywhere 10 - Error Messages*].	-172

Member name	Description	Value
SQLE_CURSOR_NOT_DECLARED	See “Cursor has not been declared” [*SQL Anywhere 10 - Error Messages*].	-170
SQLE_CURSOR_NOT_OPEN	See “Cursor not open” [*SQL Anywhere 10 - Error Messages*].	-180
SQLE_CURSOR_RESTORED	See “UltraLite cursor (or result set or table) was restored” [*SQL Anywhere 10 - Error Messages*].	134
SQLE_CURSOROP_NOT_ALLOWED	See “Illegal cursor operation attempt” [*SQL Anywhere 10 - Error Messages*].	-187
SQLE_DATABASE_ERROR	See “Internal database error %1 -- transaction rolled back” [*SQL Anywhere 10 - Error Messages*].	-301
SQLE_DATABASE_NAME_REQUIRED	See “Database name required to start server” [*SQL Anywhere 10 - Error Messages*].	-87
SQLE_DATABASE_NOT_CREATED	See “Database creation failed: %1” [*SQL Anywhere 10 - Error Messages*].	-645
SQLE_DATATYPE_NOT_ALLOWED	See “Expression has unsupported data type” [*SQL Anywhere 10 - Error Messages*].	-624
SQLE_DBSPACE_FULL	See “A dbspace has reached its maximum file size” [*SQL Anywhere 10 - Error Messages*].	-604
SQLE_DESCRIBE_NONSELECT	See “Can only describe a SELECT statement” [*SQL Anywhere 10 - Error Messages*].	-160
SQLE_DEVICE_IO_FAILED	See “File I/O failed for '%1'” [*SQL Anywhere 10 - Error Messages*].	-974
SQLE_DIV_ZERO_ERROR	See “Division by zero” [*SQL Anywhere 10 - Error Messages*].	-628
SQLE_DOWNLOAD_CONFLICT	See “Download failed because of conflicts with existing rows” [*SQL Anywhere 10 - Error Messages*].	-839
SQLE_DOWNLOAD_RESTART_FAILED	See “Unable to retry download because upload is not finished” [*SQL Anywhere 10 - Error Messages*].	-1102
SQLE_DROP_DATABASE_FAILED	See “An attempt to delete database '%1' failed” [*SQL Anywhere 10 - Error Messages*].	-651
SQLE_DUPLICATE_CURSOR_NAME	See “The cursor name '%1' already exists” [*SQL Anywhere 10 - Error Messages*].	-683
SQLE_DUPLICATE_FOREIGN_KEY	See “Foreign key '%1' for table '%2' duplicates an existing foreign key” [*SQL Anywhere 10 - Error Messages*].	-251

Member name	Description	Value
SQLE_DUPLICATE_OPTION	See “Option '%1' specified more than once” [*SQL Anywhere 10 - Error Messages*].	139
SQLE_DYNAMIC_MEMORY_EXHAUSTED	See “Dynamic memory exhausted” [*SQL Anywhere 10 - Error Messages*].	-78
SQLE_ENCRYPTION_INITIALIZATION_FAILED	See “Could not initialize the encryption DLL: '%1'” [*SQL Anywhere 10 - Error Messages*].	-984
SQLE_ENGINE_ALREADY_RUNNING	See “Database server already running” [*SQL Anywhere 10 - Error Messages*].	-96
SQLE_ERROR	See “Run time SQL error -- %1” [*SQL Anywhere 10 - Error Messages*].	-300
SQLE_ERROR_CALLING_FUNCTION	See “Could not allocate resources to call external function” [*SQL Anywhere 10 - Error Messages*].	-622
SQLE_ERROR_IN_ASSIGNMENT	See “Error in assignment” [*SQL Anywhere 10 - Error Messages*].	-641
SQLE_EXPRESSION_ERROR	See “Invalid expression near '%1'” [*SQL Anywhere 10 - Error Messages*].	-156
SQLE_FEATURE_NOT_ENABLED	See “The method you attempted to invoke was not enabled for your application” [*SQL Anywhere 10 - Error Messages*].	-1092
SQLE_FILE_BAD_DB	See “Unable to start specified database: '%1' is not a valid database file” [*SQL Anywhere 10 - Error Messages*].	-1006
SQLE_FILE_IN_USE	See “Specified database file already in use” [*SQL Anywhere 10 - Error Messages*].	-816
SQLE_FILE_NOT_DB	See “Unable to start specified database: '%1' is not a database” [*SQL Anywhere 10 - Error Messages*].	-1004
SQLE_FILE_VOLUME_NOT_FOUND	See “Specified file system volume not found for database '%1'” [*SQL Anywhere 10 - Error Messages*].	-1112
SQLE_FILE_WRONG_VERSION	See “Unable to start specified database: '%1' was created by a different version of the software” [*SQL Anywhere 10 - Error Messages*].	-1005
SQLE_FOREIGN_KEY_NAME_NOT_FOUND	See “Foreign key name '%1' not found” [*SQL Anywhere 10 - Error Messages*].	-145

Member name	Description	Value
SQLE_IDENTIFIER_TOO_LONG	See “Identifier '%1' too long” [*SQL Anywhere 10 - Error Messages*].	-250
SQLE_INCORRECT_VOLUME_ID	See “Incorrect volume ID for '%1'” [*SQL Anywhere 10 - Error Messages*].	-975
SQLE_INDEX_NAME_NOT_UNIQUE	See “Index name '%1' not unique” [*SQL Anywhere 10 - Error Messages*].	-111
SQLE_INDEX_NOT_FOUND	See “Cannot find index named '%1'” [*SQL Anywhere 10 - Error Messages*].	-183
SQLE_INDEX_NOT_UNIQUE	See “Index '%1' for table '%2' would not be unique” [*SQL Anywhere 10 - Error Messages*].	-196
SQLE_INTERRUPTED	See “Statement interrupted by user” [*SQL Anywhere 10 - Error Messages*].	-299
SQLE_INVALID_CONSTRAINT_REF	See “Invalid reference to or operation on constraint '%1'” [*SQL Anywhere 10 - Error Messages*].	-937
SQLE_INVALID_DESCRIPTOR_INDEX	See “Invalid descriptor index” [*SQL Anywhere 10 - Error Messages*].	-640
SQLE_INVALID_DESCRIPTOR_NAME	See “Invalid SQL descriptor name” [*SQL Anywhere 10 - Error Messages*].	-642
SQLE_INVALID_DISTINCT_AGGREGATE	See “Grouped query contains more than one distinct aggregate function” [*SQL Anywhere 10 - Error Messages*].	-863
SQLE_INVALID_FOREIGN_KEY	See “No primary key value for foreign key '%1' in table '%2'” [*SQL Anywhere 10 - Error Messages*].	-194
SQLE_INVALID_FOREIGN_KEY_DEF	See “Column '%1' in foreign key has a different definition than primary key” [*SQL Anywhere 10 - Error Messages*].	-113
SQLE_INVALID_GROUP_SELECT	See “Function or column reference to '%1' must also appear in a GROUP BY” [*SQL Anywhere 10 - Error Messages*].	-149
SQLE_INVALID_INDEX_TYPE	See “Index type specification of '%1' is invalid” [*SQL Anywhere 10 - Error Messages*].	-650
SQLE_INVALID_LOGON	See “Invalid user ID or password” [*SQL Anywhere 10 - Error Messages*].	-103
SQLE_INVALID_OPTION_SETTING	See “Invalid setting for option '%1'” [*SQL Anywhere 10 - Error Messages*].	-201
SQLE_INVALID_OPTION_VALUE	See “'%1' is an invalid value for '%2'” [*SQL Anywhere 10 - Error Messages*].	-1053

Member name	Description	Value
SQLE_INVALID_ORDER	See “Invalid ORDER BY specification” [*SQL Anywhere 10 - Error Messages*].	-152
SQLE_INVALID_PARAMETER	See “Invalid parameter” [*SQL Anywhere 10 - Error Messages*].	-735
SQLE_INVALID_PARSE_PARAMETER	See “Parse error: %1” [*SQL Anywhere 10 - Error Messages*].	-95
SQLE_INVALID_PUBLICATION_MASK	See “The specified publication mask is invalid” [*SQL Anywhere 10 - Error Messages*].	-1105
SQLE_INVALID_SQL_IDENTIFIER	See “Invalid SQL identifier” [*SQL Anywhere 10 - Error Messages*].	-760
SQLE_INVALID_STATEMENT	See “Invalid statement” [*SQL Anywhere 10 - Error Messages*].	-130
SQLE_INVALID_UNION	See “Select lists in UNION, INTERSECT, or EXCEPT do not match in length” [*SQL Anywhere 10 - Error Messages*].	-153
SQLE_KEYLESS_ENCRYPTION	See “Unable to perform requested operation since this database uses keyless encryption” [*SQL Anywhere 10 - Error Messages*].	-1109
SQLE_LOCKED	See “User '%1' has the row in '%2' locked” [*SQL Anywhere 10 - Error Messages*].	-210
SQLE_MEMORY_ERROR	See “Memory error -- transaction rolled back” [*SQL Anywhere 10 - Error Messages*].	-309
SQLE_METHOD_CANNOT_BE_CALLED	See “Method '%1' cannot be called at this time” [*SQL Anywhere 10 - Error Messages*].	-669
SQLE_NAME_NOT_UNIQUE	See “Item '%1' already exists” [*SQL Anywhere 10 - Error Messages*].	-110
SQLE_NO_COLUMN_NAME	See “Derived table '%1' has no name for column %2” [*SQL Anywhere 10 - Error Messages*].	-163
SQLE_NO_CURRENT_ROW	See “No current row of cursor” [*SQL Anywhere 10 - Error Messages*].	-197
SQLE_NO_INDICATOR	See “No indicator variable provided for NULL result” [*SQL Anywhere 10 - Error Messages*].	-181
SQLE_NO_MATCHING_SELECT_ITEM	See “The select list for the derived table '%1' has no expression to match '%2'” [*SQL Anywhere 10 - Error Messages*].	-812
SQLE_NO_PRIMARY_KEY	See “Table '%1' has no primary key” [*SQL Anywhere 10 - Error Messages*].	-118

Member name	Description	Value
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" [*SQL Anywhere 10 - Error Messages*].	-190
SQLE_NON_UPDATEABLE_CURSOR	See "FOR UPDATE has been incorrectly specified for a READ ONLY cursor" [*SQL Anywhere 10 - Error Messages*].	-813
SQLE_NOT_IMPLEMENTED	See "Feature '%1' not implemented" [*SQL Anywhere 10 - Error Messages*].	-134
SQLE_NOT_SUPPORTED_IN_ULTRALITE	See "Feature not available with UltraLite" [*SQL Anywhere 10 - Error Messages*].	-749
SQLE_NOTFOUND	See "Row not found" [*SQL Anywhere 10 - Error Messages*].	100
SQLE_ONLY_ONE_TABLE	See "INSERT/DELETE on cursor can modify only one table" [*SQL Anywhere 10 - Error Messages*].	-199
SQLE_OVERFLOW_ERROR	See "Value %1 out of range for destination" [*SQL Anywhere 10 - Error Messages*].	-158
SQLE_PAGE_SIZE_INVALID	See "Invalid database page size" [*SQL Anywhere 10 - Error Messages*].	-644
SQLE_PARTIAL_DOWNLOAD_NOT_FOUND	See "No partial download was found" [*SQL Anywhere 10 - Error Messages*].	-1103
SQLE_PERMISSION_DENIED	See "Permission denied: %1" [*SQL Anywhere 10 - Error Messages*].	-121
SQLE_PRIMARY_KEY_NOT_UNIQUE	See "Primary key for table '%1' is not unique" [*SQL Anywhere 10 - Error Messages*].	-193
SQLE_PRIMARY_KEY_TWICE	See "Table cannot have two primary keys" [*SQL Anywhere 10 - 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'" [*SQL Anywhere 10 - Error Messages*].	-198
SQLE_PUBLICATION_NOT_FOUND	See "Publication '%1' not found" [*SQL Anywhere 10 - Error Messages*].	-280
SQLE_PUBLICATION_PREDICATE_IGNORED	See "Publication predicates were not evaluated" [*SQL Anywhere 10 - Error Messages*].	138

Member name	Description	Value
SQLE_RESOURCE_GOVERNOR_EXCEEDED	See "Resource governor for '%1' exceeded" [*SQL Anywhere 10 - Error Messages*].	-685
SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY	See "Row was dropped from table %1 to maintain referential integrity" [*SQL Anywhere 10 - Error Messages*].	137
SQLE_ROW_EXCEEDS_PAGE_SIZE	See "A row cannot be stored because it exceeds the database page size" [*SQL Anywhere 10 - Error Messages*].	-1117
SQLE_SCHEMA_UPGRADE_NOT_ALLOWED	See "A schema upgrade is not currently allowed" [*SQL Anywhere 10 - Error Messages*].	-953
SQLE_SERVER_SYNCHRONIZATION_ERROR	See "Synchronization failed due to an error on the server: %1" [*SQL Anywhere 10 - Error Messages*].	-857
SQLE_START_STOP_DATABASE_DENIED	See "Request to start/stop database denied" [*SQL Anywhere 10 - Error Messages*].	-75
SQLE_STATEMENT_ERROR	See "SQL statement error" [*SQL Anywhere 10 - Error Messages*].	-132
SQLE_STRING_RIGHT_TRUNCATION	See "Right truncation of string data" [*SQL Anywhere 10 - Error Messages*].	-638
SQLE_SUBQUERY_SELECT_LIST	See "Subquery allowed only one select list item" [*SQL Anywhere 10 - Error Messages*].	-151
SQLE_SYNC_INFO_INVALID	See "Information for synchronization is incomplete or invalid, check '%1'" [*SQL Anywhere 10 - Error Messages*].	-956
SQLE_SYNC_INFO_REQUIRED	See "Information for synchronization was not provided" [*SQL Anywhere 10 - Error Messages*].	-1111
SQLE_SYNC_NOT_REENTRANT	See "Synchronization process was unable to re-enter synchronization" [*SQL Anywhere 10 - Error Messages*].	-1110
SQLE_SYNC_STATUS_UNKNOWN	See "The status of the last synchronization upload is unknown" [*SQL Anywhere 10 - Error Messages*].	-952
SQLE_SYNTAX_ERROR	See "Syntax error near '%1' %2" [*SQL Anywhere 10 - Error Messages*].	-131

Member name	Description	Value
SQLE_TABLE_ALREADY_INCLUDED	See “Table '%1' is already included” [*SQL Anywhere 10 - Error Messages*].	-822
SQLE_TABLE_IN_USE	See “Table in use” [*SQL Anywhere 10 - Error Messages*].	-214
SQLE_TABLE_NOT_FOUND	See “Table '%1' not found” [*SQL Anywhere 10 - Error Messages*].	-141
SQLE_TOO_MANY_BLOB_REFS	See “Too many references to a BLOB” [*SQL Anywhere 10 - Error Messages*].	-1107
SQLE_TOO_MANY_CONNECTIONS	See “Database server connection limit exceeded” [*SQL Anywhere 10 - Error Messages*].	-102
SQLE_TOO_MANY_PUBLICATIONS	See “Too many publications specified in publication mask” [*SQL Anywhere 10 - Error Messages*].	-1106
SQLE_TOO_MANY_TEMP_TABLES	See “Too many temporary tables in connection” [*SQL Anywhere 10 - Error Messages*].	-817
SQLE_TOO_MANY_USERS	See “Too many users in database” [*SQL Anywhere 10 - Error Messages*].	-1104
SQLE_ULTRALITE_DATABASE_NOT_FOUND	See “The database '%1' was not found” [*SQL Anywhere 10 - Error Messages*].	-954
SQLE_ULTRALITE_OBJ_CLOSED	See “Invalid operation on a closed object” [*SQL Anywhere 10 - Error Messages*].	-908
SQLE_ULTRALITE_WRITE_ACCESS_DENIED	See “Write access was denied” [*SQL Anywhere 10 - Error Messages*].	-1086
SQLE_UNABLE_TO_CONNECT	See “Database cannot be started -- %1” [*SQL Anywhere 10 - Error Messages*].	-105
SQLE_UNABLE_TO_START_DATABASE	See “Unable to start specified database: %1” [*SQL Anywhere 10 - Error Messages*].	-82
SQLE_UNABLE_TO_START_DATABASE_VER_NEWER	See “Unable to start specified database: Server must be upgraded to start database %1” [*SQL Anywhere 10 - Error Messages*].	-934
SQLE_UNCOMMITTED_TRANSACTIONS	See “You cannot synchronize or upgrade with uncommitted transactions” [*SQL Anywhere 10 - Error Messages*].	-755
SQLE_UNKNOWN_FUNC	See “Unknown function '%1'” [*SQL Anywhere 10 - Error Messages*].	-148

Member name	Description	Value
SQLE_UNKNOWN_OPTION	See "'%1' is an unknown option" [*SQL Anywhere 10 - Error Messages*].	120
SQLE_UNKNOWN_USERID	See "User ID '%1' does not exist" [*SQL Anywhere 10 - Error Messages*].	-140
SQLE_UNRECOGNIZED_OPTION	See "The option '%1' is not recognized" [*SQL Anywhere 10 - Error Messages*].	-1002
SQLE_UPLOAD_FAILED_AT_SERVER	See "Synchronization server failed to commit the upload" [*SQL Anywhere 10 - Error Messages*].	-794
SQLE_VALUE_IS_NULL	See "Cannot return NULL result as requested data type" [*SQL Anywhere 10 - Error Messages*].	-1050
SQLE_VARIABLE_INVALID	See "Invalid host variable" [*SQL Anywhere 10 - Error Messages*].	-155
SQLE_WRONG_NUM_OF_INSERT_COLS	See "Wrong number of values for INSERT" [*SQL Anywhere 10 - Error Messages*].	-207
SQLE_WRONG_PARAMETER_COUNT	See "Wrong number of parameters to function '%1'" [*SQL Anywhere 10 - Error Messages*].	-154

ULStreamErrorCode enumeration

UL Ext.: Enumerates the error codes that may be reported by streams during synchronization.

Prototypes

Visual Basic
Public Enum **ULStreamErrorCode**

C#
public enum **ULStreamErrorCode**

Members

Member name	Description	Value
ACTSYNC_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_INSTALLED	The ActiveSync provider has not been installed. Run dbasinst to install it (see documentation for details).	76
CONNECT_TIMEOUT	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_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_RANDOM_OBJECT	The secure network layer could not create a random-number-generating object. Free up system resources, reconnect and retry the operation.	17
DEQUEUING_CONNECTION	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_SUPPORTED	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 above. Dialup is not supported on other platforms.	203

Member name	Description	Value
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_RANDOM	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_AUTHENTICATION_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_AUTHENTICATION_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_STATUS_CODE	Examine the status line to determine the cause of the failure.	86
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

Member name	Description	Value
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
HTTP_INVALID_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_MALFORMED_SESSION_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_PASSWORD	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_AUTHENTICATION_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_AUTHENTICATION_REQUIRED	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_SERVER_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_UNABLE_TO_PARSE_COOKIE	Determine where the set cookie header is being corrupted.	88

Member name	Description	Value
HTTP_UNKNOWN_TRANSFER_ENCODING	Determine how the unknown transfer encoding is getting generated.	87
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-complient (ie. 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_SYNC_PROTOCOL	The specified protocol is not a valid synchronization protocol.	226
LIBRARY_ENTRY_POINT_NOT_FOUND	The indicated library entry point could not be found.	225
LOAD_LIBRARY_FAILURE	The indicated library could not be found in the path. In you are trying to use TLS encryption for synchronization, make sure you have acquired the proper license.	224

Member name	Description	Value
LOAD_NETWORK_LIBRARY	The network interface library could not be found and/or loaded. Please check the following: 1) The sockets layer is properly installed. The correct 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_ALLOCATION	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 operating system documentation for details.	6
MISSING_PARAMETER	The specified parameter was expected but not supplied.	229
NO_ECC_FIPS	There was a problem performing the given compression operation. Please contact technical support.	239
NONE	This code indicates there was either no network error, or an unknown network error occurred.	0
NOT_IMPLEMENTED	An unimplemented internal feature was requested. Please contact technical support.	12
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

Member name	Description	Value
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
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 output 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 output 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

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_CHAIN_FUNC_XXX	Unused.	28
SECURE_CERTIFICATE_CHAIN_LENGTH_XXX	Unused.	22
SECURE_CERTIFICATE_CHAIN_REF_XXX	Unused.	29
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 readcert 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 readcert 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 readcert utility.	51

Member name	Description	Value
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
SECURE_CERTIFICATE_EXPIRED	A certificate in the certificate chain has expired. Obtain a new certificate with a later expiry date and retry the operation.	50
SECURE_CERTIFICATE_EXPIRY_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 certificates. 3) The certificate file contains the correct certificate (s). 4) The certificate file is undamaged.	37
SECURE_CERTIFICATE_FILE_NOT_FOUND	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 certificates. 4) The certificate file contains the correct certificate (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

Member name	Description	Value
SECURE_CERTIFICATE_NOT_TRUSTED	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 certificates. 3) The certificate file contains the correct certificate (s). 4) The client's list of trusted root certificates includes the server's root certificate.	24
SECURE_CERTIFICATE_REF_XXX	Unused.	23
SECURE_CERTIFICATE_ROOT	The root certificate in the chain is invalid. At the time of publication, this error was defined but not used.	20
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_ENABLE_NON_BLOCKING_XXX	Unused.	30
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 certificate file was specified, both on the client and on the server.	53

Member name	Description	Value
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
SECURE_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 as well as 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

Member name	Description	Value
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
SECURE_REDUNDANT_SERVER_CERTIFICATE_PASSWORD	A password was specified when the server's private key wasn't encrypted by any password.	208
SECURE_SET_CHAIN_NUMBER_XXX	Unused.	32
SECURE_SET_CIPHER_SUITES_XXX	Unused.	31
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_IO_SEMANTICS_XXX	An internal error has occurred in the network layer. Please contact technical support.	27
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_SET_PROTOCOL_SIDE_XXX	Unused.	47
SECURE_SET_RANDOM_FUNC_XXX	Unused.	46
SECURE_SET_RANDOM_REF_XXX	Unused.	45

Member name	Description	Value
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
SECURE_TRUSTED_CERTIFICATE_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 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.	41
SEED_RANDOM	The secure network layer could not seed its random number generator. Free up system resources, reconnect 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 (eg. 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_CREATE_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 (eg. 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
SOCKET_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
SOCKET_GET_NAME	The network layer was unable to determine a socket'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 connection 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 normally. 3) There are enough system resources available. Free up system resources if they are running low.	64
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

Member name	Description	Value
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 (eg. 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
SOCKET_LOCALHOST_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
SOCKET_PORT_OUT_OF_RANGE	An invalid port number was specified. The port number must be an integer between zero and 65535.	74

Member name	Description	Value
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_OPTION	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
SOCKET_SHUTDOWN	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
UNEXPECTED_HTTP_REQUEST_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
UNRECOGNIZED_TLS_TYPE	The TLS type is invalid. Consult the documentation for valid types.	240

Member name	Description	Value
VALUE_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
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 output 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 output 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 826

ULStreamErrorContext enumeration

UL Ext.: Enumerates the basic network operation being performed when the stream errors occured.

Prototypes

Visual Basic
Public Enum **ULStreamErrorContext**

C#
public enum **ULStreamErrorContext**

Members

Member name	Description	Value
CLOSE	CLOSE(6)	6
CREATE	CREATE(3)	3
DESTROY	DESTROY(4)	4
END_READ	END_READ(11)	11
END_WRITE	END_WRITE(10)	10
GETVALUE	GETVALUE(14)	14
OPEN	OPEN(5)	5
READ	READ(7)	7
REGISTER	REGISTER(1)	1
SOFTSHUTDOWN	SOFTSHUTDOWN(13)	13
UNKNOWN	UNKNOWN(0)	0
UNREGISTER	UNREGISTER(2)	2
WRITE	WRITE(8)	8
WRITE_FLUSH	WRITE_FLUSH(9)	9
YIELD	YIELD(12)	12

See also

- "StreamErrorContext property" on page 827

ULStreamErrorID enumeration

UL Ext.: Enumerates the network layers that may report errors during synchronization.

Prototypes

Visual Basic
Public Enum **ULStreamErrorID**

C#
public enum **ULStreamErrorID**

Members

Member name	Description	Value
ACTIVESYNC	ACTIVESYNC(22)	22
CERTICOM	CERTICOM(11)	11
CERTICOM_SSL	CERTICOM_SSL(13)	13
CERTICOM_TLS	CERTICOM_TLS(14)	14
DH_CAST	DH_CAST(9)	9
EMAIL	EMAIL(20)	20
FAKE	FAKE(2)	2
FILE	FILE(21)	21
HTTP	HTTP(7)	7
HTTPS	HTTPS(8)	8
JAVA_CERTICOM	JAVA_CERTICOM(12)	12
JAVA_RSA	JAVA_RSA(24)	24
NETTECH	NETTECH(5)	5
OPEN_SSL_RSA	OPEN_SSL_RSA(25)	25
PALM_CONDUIT	PALM_CONDUIT(3)	3
PALM_SS	PALM_SS(4)	4
PALM_SSL	PALM_SSL(26)	26
REPLAY	REPLAY(17)	17
RIMBB	RIMBB(6)	6

Member name	Description	Value
RSA_TLS	RSA_TLS(23)	23
SECURE	SECURE(10)	10
SERIAL	SERIAL(1)	1
STRM	STRM(18)	18
TCPIP	TCPIP(0)	0
UDP	UDP(19)	19
WIRELESS	WIRELESS(16)	16
WIRESTRM	WIRESTRM(15)	15

See also

- "StreamErrorID property" on page 827

ULStreamType enumeration

UL Ext.: Enumerates the types of MobiLink synchronization streams to use for synchronization.

Prototypes

Visual Basic
Public Enum **ULStreamType**

C#
public enum **ULStreamType**

Remarks

For information on configuring specific stream types, see "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*].

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). **Separately licensed component required** ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations. See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].	2
TCPIP	Synchronize via TCP/IP.	0
TLS	Synchronize via TCP/IP with transport layer security. **Separately licensed component required** ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations. See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].	3

See also

- "Stream property" on page 805

ULSyncParms class

UL Ext.: Represents synchronization parameters that define how to synchronize an UltraLite database. This class cannot be inherited.

Prototypes

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 "SyncParms property" on page 509.

At most, only one synchronization command ("DownloadOnly property" on page 799, "PingOnly property" on page 802, "ResumePartialDownload property" on page 803, or "UploadOnly property" on page 807) can be specified at a time. If more than one of these parameters is set to true, a "ULSQLCode enumeration" on page 763 SQLException is thrown by "Synchronize() method" on page 527.

Other sources of "ULSQLCode enumeration" on page 763 errors include not specifying a "Stream property" on page 805 value or a "Version property" on page 808 value.

See also

- "ULSyncParms members" on page 796
- "ULConnection class" on page 498
- "SyncParms property" on page 509
- "Synchronize() method" on page 527

ULSyncParms members

Public properties

Member name	Description
AuthenticationParms property	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
CheckpointStore property	Specifies whether the client should perform extra store checkpoints to control the growth of the database store during synchronization.
DisableConcurrency property	Specifies whether concurrent access to UltraLite while performing a synchronization.
DownloadOnly property	Specifies whether to disable or enable uploads when synchronizing.

Member name	Description
KeepPartialDownload property	Specifies whether to disable or enable partial downloads when synchronizing.
NewPassword property	Specifies a new MobiLink password for the user specified with UserName.
Password property	The MobiLink password for the user specified by UserName.
PingOnly property	Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.
PublicationMask property	Specifies the publications to be synchronized.
ResumePartialDownload property	Specifies whether to resume or discard a previous partial download.
SendColumnNames property	Specifies whether the client should send column names to the MobiLink server during synchronization.
SendDownloadAck property	Specifies whether the client should send a download acknowledgement to the MobiLink server during synchronization. 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	Specifies the MobiLink synchronization stream to use for synchronization.
StreamParms property	Specifies the parameters to configure the synchronization stream.
TableOrder property	Specifies the order tables should be uploaded to the consolidated database.
UploadOnly property	Specifies whether to disable or enable downloads when synchronizing.
UserName property	The user name that uniquely identifies the MobiLink client to the MobiLink server.
Version property	Specifies which synchronization script to use.

Public methods

Member name	Description
CopyFrom method	Copies the properties of the specified “ULSyncParms class” on page 796 object to this “ULSyncParms class” on page 796 object.

See also

- “ULSyncParms class” on page 796
- “ULConnection class” on page 498

- "SyncParms property" on page 509
- "Synchronize() method" on page 527

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Prototypes

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 796
- "ULSyncParms members" on page 796

CheckpointStore property

Specifies whether the client should perform extra store checkpoints to control the growth of the database store during synchronization.

Prototypes

Visual Basic
Public Property **CheckpointStore** As Boolean

C#
public bool **CheckpointStore** { get; set; }

Property value

True to specify that the client should perform extra store checkpoints. The default is false, meaning only required checkpointing is done.

Remarks

The checkpoint operation adds I/O operations for the application, and so slows synchronization. This option is most useful for large downloads with many updates. Devices with slow flash memory might not want to pay the performance penalty associated with additional checkpoints.

See also

- "ULSyncParms class" on page 796
- "ULSyncParms members" on page 796

DisableConcurrency property

Specifies whether concurrent access to UltraLite while performing a synchronization.

Prototypes

Visual Basic
Public Property **DisableConcurrency** As Boolean

C#
public bool **DisableConcurrency** { get; set; }

Property value

True to disable concurrent access to UltraLite while performing a synchronization, false to enable concurrent access. The default is false.

Remarks

By default, other threads can perform UltraLite operations while a thread is synchronizing. When concurrent synchronization is disabled, other threads block on UltraLite calls until synchronization completes.

See also

- "ULSyncParms class" on page 796
- "ULSyncParms members" on page 796

DownloadOnly property

Specifies whether to disable or enable uploads when synchronizing.

Prototypes

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 ("DownloadOnly property" on page 799, "PingOnly property" on page 802, "ResumePartialDownload property" on page 803, or "UploadOnly property" on page 807) can be specified at a time. If more than one of these parameters is set to true, a "ULSQLCode enumeration" on page 763 SQLException is thrown by "Synchronize() method" on page 527.

See also

- "ULSyncParms class" on page 796
- "ULSyncParms members" on page 796
- "UploadOnly property" on page 807

KeepPartialDownload property

Specifies whether to disable or enable partial downloads when synchronizing.

Prototypes

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."PartialDownloadRetained property" on page 826 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."ResumePartialDownload property" on page 803 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 PublicationMask 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 property" on page 803 and "UserName property" on page 807. The fields KeepPartialDownload and "DisableConcurrency property" on page 799 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 "RollbackPartialDownload method" on page 526 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 796
- "ULSyncParms members" on page 796
- "PartialDownloadRetained property" on page 826
- "ResumePartialDownload property" on page 803
- "RollbackPartialDownload method" on page 526

NewPassword property

Specifies a new MobiLink password for the user specified with UserName.

Prototypes

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 796
- "ULSyncParms members" on page 796
- "UserName property" on page 807

Password property

The MobiLink password for the user specified by UserName.

Prototypes

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 796
- "ULSyncParms members" on page 796
- "NewPassword property" on page 801
- "UserName property" on page 807

PingOnly property

Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.

Prototypes

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 ("DownloadOnly property" on page 799, "PingOnly property" on page 802, "ResumePartialDownload property" on page 803, or "UploadOnly property" on page 807) can be specified at a time. If more than one of these parameters is set to true, a "ULSQLCode enumeration" on page 763 SQLException is thrown by "Synchronize() method" on page 527.

See also

- "ULSyncParms class" on page 796
- "ULSyncParms members" on page 796

PublicationMask property

Specifies the publications to be synchronized.

Prototypes

Visual Basic
Public Property **PublicationMask** As Integer

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

Property value

A bitwise combination of publication masks, the special value "SYNC_ALL_PUBS field" on page 721, or the special value "SYNC_ALL_DB field" on page 721. The default is "SYNC_ALL_DB field" on page 721. For more information on publication masks, see "ULPublicationSchema class" on page 720.

See also

- "ULSyncParms class" on page 796
- "ULSyncParms members" on page 796
- "Mask property" on page 722

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Prototypes

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 ("DownloadOnly property" on page 799, "PingOnly property" on page 802, "ResumePartialDownload property" on page 803, or "UploadOnly property" on page 807) can be specified at a time. If more than one of these parameters is set to true, a "ULSQLCode enumeration" on page 763 SQLException is thrown by "Synchronize() method" on page 527.

For more information on partial downloads, see "KeepPartialDownload property" on page 800.

See also

- "ULSyncParms class" on page 796
- "ULSyncParms members" on page 796

- "PartialDownloadRetained property" on page 826

SendColumnNames property

Specifies whether the client should send column names to the MobiLink server during synchronization.

Prototypes

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

This parameter is typically used together with the -za or -ze option on the MobiLink server for automatically generating synchronization scripts.

See also

- "ULSyncParms class" on page 796
- "ULSyncParms members" on page 796

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).

Prototypes

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 796
- "ULSyncParms members" on page 796

Stream property

Specifies the MobiLink synchronization stream to use for synchronization.

Prototypes

Visual Basic
Public Property **Stream** As ULStreamType

C#
public ULStreamType **Stream** { get; set; }

Property value

One of the "ULStreamType enumeration" on page 794 values specifying the type of synchronization stream to use. The default is "ULStreamType enumeration" on page 794.

Remarks

Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. These parameters are supplied by the "StreamParms property" on page 805.

If the stream type is set to a value that is invalid for the platform, the stream type is set to "ULStreamType enumeration" on page 794.

See also

- "ULSyncParms class" on page 796
- "ULSyncParms members" on page 796
- "ULStreamType enumeration" on page 794
- "StreamParms property" on page 805

StreamParms property

Specifies the parameters to configure the synchronization stream.

Prototypes

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 on configuring specific stream types, see "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*].

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 796
- "ULSyncParms members" on page 796
- "Stream property" on page 805
- "ULStreamType enumeration" on page 794

TableOrder property

Specifies the order tables should be uploaded to the consolidated database.

Prototypes

Visual Basic
Public Property **TableOrder** As String

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

Property value

A string, in the form of a comma-separated list of table names. Tables names can be quoted using either single or double quotes. The default is a null reference (Nothing in Visual Basic), which does not override the default ordering of tables.

Remarks

If the foreign keys on your consolidated database match the foreign keys on your remote UltraLite database and there are no foreign key cycles, then you most likely don't need to use this feature. However, if you have tables that are part of foreign key cycles then list all tables that are part of a cycle in the TableOrder field. If you have tables with different foreign key relationships on the consolidated then also list these tables in the TableOrder field.

All tables that you don't specify will be appropriately sorted based of the foreign keys defined in the remote database.

See also

- "ULSyncParms class" on page 796
- "ULSyncParms members" on page 796

UploadOnly property

Specifies whether to disable or enable downloads when synchronizing.

Prototypes

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 ("DownloadOnly property" on page 799, "PingOnly property" on page 802, "ResumePartialDownload property" on page 803, or "UploadOnly property" on page 807) can be specified at a time. If more than one of these parameters is set to true, a "ULSQLCode enumeration" on page 763 SQLException is thrown by "Synchronize() method" on page 527.

See also

- "ULSyncParms class" on page 796
- "ULSyncParms members" on page 796
- "DownloadOnly property" on page 799

UserName property

The user name that uniquely identifies the MobiLink client to the MobiLink server.

Prototypes

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 796
- "ULSyncParms members" on page 796
- "Password property" on page 801

Version property

Specifies which synchronization script to use.

Prototypes

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 796
- "ULSyncParms members" on page 796

CopyFrom method

Copies the properties of the specified "ULSyncParms class" on page 796 object to this "ULSyncParms class" on page 796 object.

Prototypes

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 796
- “ULSyncParms members” on page 796

ULSyncProgressData class

UL Ext.: Returns synchronization progress monitoring data.

Prototypes

Visual Basic
Public Class **ULSyncProgressData**

C#
public class **ULSyncProgressData**

See also

- "ULSyncProgressData members" on page 810
- "ULSyncProgressListener interface" on page 820

ULSyncProgressData members

Public fields

Member name	Description
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.

Public properties

Member name	Description
ErrorMessage property	Returns the error message describing the error that occurred during synchronization.
Flags property	Returns the current synchronization flags indicating additional information relating to the current state.
ReceivedBytes property	Returns the number of bytes received so far. This information is updated for all states.
ReceivedDeletes property	Returns the number of deleted rows received so far.
ReceivedInserts property	Returns the number of inserted rows received so far.
ReceivedUpdates property	Returns the number of updated rows received so far.
SQLCode property	Returns the SQL code for synchronization.
SentBytes property	Returns the number of bytes sent so far. This information is updated for all states.
SentDeletes property	Returns the number of deleted rows sent so far.

Member name	Description
SentInserts property	Returns the number of inserted rows sent so far.
SentUpdates property	Returns the number of updated rows sent so far.
State property	Returns the current synchronization state.
SyncParms property	Returns a reference to the connection's SyncParms object.
SyncResult property	Returns a reference to the connection's SyncResult object. This object is only updated with "ULSyncProgressState enumeration" on page 822 and "ULSyncProgressState enumeration" on page 822 events.
TableCount property	A count of the tables sent or received (TableCount of TableTotal) so far.
TableIndex property	Returns the index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).
TableTotal property	Returns the number of tables being synchronized.

See also

- "ULSyncProgressData class" on page 810
- "ULSyncProgressListener interface" on page 820

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.

Prototypes

Visual Basic
Public Shared **FLAG_IS_BLOCKING** As Integer

C#
public const int **FLAG_IS_BLOCKING** ;

See also

- "ULSyncProgressData class" on page 810
- "ULSyncProgressData members" on page 810

ErrorMessage property

Returns the error message describing the error that occurred during synchronization.

Prototypes

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

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

Property value

A string describing the error that occurred during synchronization.

See also

- "ULSyncProgressData class" on page 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822

Flags property

Returns the current synchronization flags indicating additional information relating to the current state.

Prototypes

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 810
- "ULSyncProgressData members" on page 810
- "FLAG_IS_BLOCKING field" on page 811

ReceivedBytes property

Returns the number of bytes received so far. This information is updated for all states.

Prototypes

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 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

ReceivedDeletes property

Returns the number of deleted rows received so far.

Prototypes

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 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

ReceivedInserts property

Returns the number of inserted rows received so far.

Prototypes

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 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

ReceivedUpdates property

Returns the number of updated rows received so far.

Prototypes

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 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

SQLCode property

Returns the SQL code for synchronization.

Prototypes

Visual Basic
Public Readonly Property **SQLCode** As ULSQLCode

C#
public ULSQLCode **SQLCode** { get;}

Property value

The "ULSQLCode enumeration" on page 763 value for any synchronization error.

See also

- "ULSyncProgressData class" on page 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822

SentBytes property

Returns the number of bytes sent so far. This information is updated for all states.

Prototypes

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 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

SentDeletes property

Returns the number of deleted rows sent so far.

Prototypes

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 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

SentInserts property

Returns the number of inserted rows sent so far.

Prototypes

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 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

SentUpdates property

Returns the number of updated rows sent so far.

Prototypes

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 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

State property

Returns the current synchronization state.

Prototypes

Visual Basic
Public Readonly Property **State** As ULSyncProgressState

C#
public ULSyncProgressState **State** { get;}

Property value

One of the "ULSyncProgressState enumeration" on page 822 values specifying the current synchronization state.

See also

- "ULSyncProgressData class" on page 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822

SyncParms property

Returns a reference to the connection's SyncParms object.

Prototypes

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

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

Property value

A reference to the "SyncParms property" on page 509 object.

See also

- "ULSyncProgressData class" on page 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822
- "SyncParms property" on page 509

SyncResult property

Returns a reference to the connection's SyncResult object. This object is only updated with "ULSyncProgressState enumeration" on page 822 and "ULSyncProgressState enumeration" on page 822 events.

Prototypes

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

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

Property value

A reference to the "SyncResult property" on page 509 object.

See also

- "ULSyncProgressData class" on page 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

TableCount property

A count of the tables sent or received (TableCount of TableTotal) so far.

Prototypes

Visual Basic
Public Readonly Property **TableCount** As Integer

C#
public int **TableCount** { get;}

Property value

The count of tables sent or received.

See also

- "ULSyncProgressData class" on page 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

TableIndex property

Returns the index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).

Prototypes

Visual Basic
Public Readonly Property **TableIndex** As Integer

C#
public int **TableIndex** { get;}

Property value

The index of the table currently being synchronized. Tables are numbered 1 to "TableCount property" on page 595.

See also

- "ULSyncProgressData class" on page 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

TableTotal property

Returns the number of tables being synchronized.

Prototypes

Visual Basic
Public Readonly Property **TableTotal** As Integer

C#
public int **TableTotal** { get;}

Property value

The number of tables being synchronized.

See also

- "ULSyncProgressData class" on page 810
- "ULSyncProgressData members" on page 810
- "ULSyncProgressState enumeration" on page 822
- "ULSyncProgressState enumeration" on page 822

ULSyncProgressListener interface

UL Ext.: The listener interface for receiving synchronization progress events.

Prototypes

Visual Basic
Public Interface **ULSyncProgressListener**

C#
public interface **ULSyncProgressListener**

See also

- "ULSyncProgressListener members" on page 820
- "Synchronize(ULSyncProgressListener) method" on page 527

ULSyncProgressListener members

Public methods

Member name	Description
SyncProgressed method	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 820
- "Synchronize(ULSyncProgressListener) method" on page 527

SyncProgressed method

Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue.

Prototypes

Visual Basic
```
Public Function SyncProgressed( _
  ByVal data As ULSyncProgressData _
) As Boolean
```

C#
```
public bool SyncProgressed(
  ULSyncProgressData data
);
```


Parameters

- **data** A "ULSyncProgressData class" on page 810 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 820
- "ULSyncProgressListener members" on page 820

ULSyncProgressState enumeration

UL Ext.: Enumerates all the states that can occur while synchronizing.

Prototypes

Visual Basic
Public Enum **ULSyncProgressState**

C#
public enum **ULSyncProgressState**

Members

Member name	Description	Value
STATE_CANCELLED	Synchronization has been cancelled.	15
STATE_COMMITTING_DOWNLOAD	The download is being committed. The final count of rows received is included with this event. See "ReceivedBytes property" on page 812, "ReceivedInserts property" on page 813, "ReceivedUpdates property" on page 814, and "ReceivedDeletes property" on page 813.	9
STATE_CONNECTING	The synchronization stream has been built, but is not yet opened.	1
STATE_DISCONNECTING	The synchronization stream is about to be closed.	11
STATE_DONE	Synchronization has successfully completed. The connection's "SyncResult property" on page 817 object has been updated.	12
STATE_ERROR	Synchronization has completed, but an error occurred. Check "SyncResult property" on page 817, "ErrorMessage property" on page 811, and "SQLCode property" on page 814 for details.	13
STATE_FINISHING_UPLOAD	The upload is completing. The final count of rows sent is included with this event. See "SentBytes property" on page 814, "SentInserts property" on page 815, "SentUpdates property" on page 816, and "SentDeletes property" on page 815.	5
STATE_LAST	The last state entered by synchronization. This state is always entered and always after any other state. Other ULSyncProgressData fields may not contain valid data.	128

Member name	Description	Value
STATE_RECEIVING_DATA	Data for the current table is being received. "ReceivedBytes property" on page 812, "ReceivedInserts property" on page 813, "ReceivedUpdates property" on page 814, and "ReceivedDeletes property" on page 813 have been updated.	8
STATE_RECEIVING_TABLE	A table is being received. Progress can be monitored using "TableIndex property" on page 818 and "TableCount property" on page 817.	7
STATE_RECEIVING_UPLOAD_ACK	An acknowledgement that the upload is complete is being received.	6
STATE_ROLLING_BACK_DOWNLOAD	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_SENDING_DATA	Data for the current table is being sent. "SentBytes property" on page 814, "SentInserts property" on page 815, "SentUpdates property" on page 816, and "SentDeletes property" on page 815 have been updated.	4
STATE_SENDING_DOWNLOAD_ACK	An acknowledgement that the download is complete is being sent.	10
STATE_SENDING_HEADER	The synchronization stream has been opened and the header is about to be sent.	2
STATE_SENDING_TABLE	A table is being sent. Progress can be monitored using "TableIndex property" on page 818 and "TableCount property" on page 817.	3
STATE_STARTING	No synchronization actions have been taken yet.	0

See also

- "ULSyncProgressData class" on page 810

ULSyncResult class

UL Ext.: Represents the status of the last synchronization.

Prototypes

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 "SyncResult property" on page 509. A ULSyncResult instance is only valid while that connection is open.

See also

- "ULSyncResult members" on page 824
- "SyncResult property" on page 509
- "Synchronize() method" on page 527

ULSyncResult members

Public properties

Member name	Description
AuthStatus property	Returns the authorization status code for the last synchronization attempt.
AuthValue property	Returns the return value from custom user authentication synchronization scripts.
IgnoredRows property	Checks whether any uploaded rows were ignored during the last synchronization.
PartialDownloadRetained property	Checks whether a partial download was retained during the last synchronization.
StreamErrorCode property	Returns the error reported by the stream itself.
StreamErrorContext property	Returns the basic network operation being performed when the stream error occurred.
StreamErrorID property	Returns the ID of the network layer reporting an error.
StreamErrorSystem property	Returns the stream error system-specific code.
Timestamp property	Returns the timestamp of the last synchronization.
UploadOK property	Checks whether the last upload synchronization was successful.

See also

- "ULSyncResult class" on page 824
- "SyncResult property" on page 509
- "Synchronize() method" on page 527

AuthStatus property

Returns the authorization status code for the last synchronization attempt.

Prototypes

Visual Basic
Public Readonly Property **AuthStatus** As ULAuthStatusCode

C#
public ULAuthStatusCode **AuthStatus** { get;}

Property value

One of the "ULAuthStatusCode enumeration" on page 433 values denoting the authorization status for the last synchronization attempt.

See also

- "ULSyncResult class" on page 824
- "ULSyncResult members" on page 824

AuthValue property

Returns the return value from custom user authentication synchronization scripts.

Prototypes

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 824
- "ULSyncResult members" on page 824

IgnoredRows property

Checks whether any uploaded rows were ignored during the last synchronization.

Prototypes

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 824
- "ULSyncResult members" on page 824
- "DownloadOnly property" on page 799

PartialDownloadRetained property

Checks whether a partial download was retained during the last synchronization.

Prototypes

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 824
- "ULSyncResult members" on page 824
- "KeepPartialDownload property" on page 800

StreamErrorCode property

Returns the error reported by the stream itself.

Prototypes

Visual Basic
Public Readonly Property **StreamErrorCode** As ULStreamErrorCode

C#
public ULStreamErrorCode **StreamErrorCode** { get;}

Property value

One of the "ULStreamErrorCode enumeration" on page 773 values denoting the error reported by the stream itself, "ULStreamErrorCode enumeration" on page 773 if no error occurred.

See also

- "ULSyncResult class" on page 824
- "ULSyncResult members" on page 824

StreamErrorContext property

Returns the basic network operation being performed when the stream error occurred.

Prototypes

Visual Basic
Public Readonly Property **StreamErrorContext** As ULStreamErrorContext

C#
public ULStreamErrorContext **StreamErrorContext** { get;}

Property value

One of the "ULStreamErrorContext enumeration" on page 791 values denoting which basic network operation was being performed when the stream error occurred.

See also

- "ULSyncResult class" on page 824
- "ULSyncResult members" on page 824

StreamErrorID property

Returns the ID of the network layer reporting an error.

Prototypes

Visual Basic
Public Readonly Property **StreamErrorID** As ULStreamErrorID

C#
public ULStreamErrorID **StreamErrorID** { get;}

Property value

One of the "ULStreamErrorID enumeration" on page 792 values denoting the ID of the network layer reporting an error.

See also

- "ULSyncResult class" on page 824
- "ULSyncResult members" on page 824

StreamErrorSystem property

Returns the stream error system-specific code.

Prototypes

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 824
- "ULSyncResult members" on page 824

Timestamp property

Returns the timestamp of the last synchronization.

Prototypes

Visual Basic
Public Readonly Property **Timestamp** As Date

C#
public DateTime **Timestamp** { get;}

Property value

A DateTime specifying the timestamp of the last synchronization.

See also

- "ULSyncResult class" on page 824
- "ULSyncResult members" on page 824

UploadOK property

Checks whether the last upload synchronization was successful.

Prototypes

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 824
- "ULSyncResult members" on page 824

ULTable class

UL Ext.: Represents a table in an UltraLite database.

Prototypes

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 "ExecuteTable() method" on page 489 of the "ULCommand class" on page 462.

Inherits: "ULResultSet class" on page 724

Implements: IDataReader, IDataRecord, IDisposable

See also

- "ULTable members" on page 830

ULTable members

Public properties

Member name	Description
Depth property (inherited from ULDataReader)	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
FieldCount property (inherited from ULDataReader)	Returns the number of columns in the cursor.
HasRows property (inherited from ULDataReader)	Checks whether the ULDataReader has one or more rows.
IsBOF property (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is before the first row.
IsClosed property (inherited from ULDataReader)	Checks whether the cursor is currently open.
IsEOF property (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is after the last row.
Item properties (inherited from ULDataReader)	Gets the value of a specified column as an instance of Object.

Member name	Description
RecordsAffected property (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 (inherited from ULDataReader)	**UL Ext.:** Returns the number of rows in the cursor.
Schema property	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 (inherited from ULResultSet)	Appends the specified subset of the specified array of Bytes to the new value for the specified "ULDbType enumeration" on page 637 column.
AppendChars method (inherited from ULResultSet)	Appends the specified subset of the specified array of Chars to the new value for the specified "ULDbType enumeration" on page 637 column.
Close method (inherited from ULDataReader)	Closes the cursor.
Delete method (inherited from ULResultSet)	Deletes the current row.
DeleteAllRows method	Deletes all rows in the table.
Dispose (inherited from DbDataReader)	Releases the resources consumed by this DbDataReader.
FindBegin method	Prepares to perform a new Find on a table.
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.
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.
FindNext methods	Continues a "FindFirst() method" on page 837 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.

Member name	Description
FindPrevious methods	Continues a "FindLast() method" on page 839 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 (inherited from ULDataReader)	Returns the value for the specified column as a Boolean.
GetByte method (inherited from ULDataReader)	Returns the value for the specified column as an unsigned 8-bit value (Byte).
GetBytes methods (inherited from ULDataReader)	**UL Ext.:** Returns the value for the specified column as an array of Bytes. Only valid for columns of type "ULDbType enumeration" on page 637, "ULDbType enumeration" on page 637, or "ULDbType enumeration" on page 637.
GetChar method (inherited from ULDataReader)	This method is not supported in UltraLite.NET.
GetChars method (inherited from ULDataReader)	Copies a subset of the value for the specified "ULDbType enumeration" on page 637 column, beginning at the specified offset, to the specified offset of the destination Char array.
GetData (inherited from DbDataReader)	Returns a DbDataReader object for the requested column ordinal.
GetDataTypeName method (inherited from ULDataReader)	Returns the name of the specified column's provider data type.
GetDateTime method (inherited from ULDataReader)	Returns the value for the specified column as a DateTime with millisecond accuracy.
GetDecimal method (inherited from ULDataReader)	Returns the value for the specified column as a Decimal.
GetDouble method (inherited from ULDataReader)	Returns the value for the specified column as a Double.
GetEnumerator method (inherited from ULDataReader)	Returns an IEnumerator that iterates through the ULDataReader.
GetFieldType method (inherited from ULDataReader)	Returns the Type most appropriate for the specified column.
GetFloat method (inherited from ULDataReader)	Returns the value for the specified column as a Single.
GetGuid method (inherited from ULDataReader)	Returns the value for the specified column as a UUID (Guid).
GetInt16 method (inherited from ULDataReader)	Returns the value for the specified column as an Int16.

Member name	Description
GetInt32 method (inherited from ULDataReader)	Returns the value for the specified column as an Int32.
GetInt64 method (inherited from ULDataReader)	Returns the value for the specified column as an Int64.
GetName method (inherited from ULDataReader)	Returns the name of the specified column.
GetOrdinal method (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.
GetSchemaTable method (inherited from ULDataReader)	Returns a DataTable that describes the column metadata of the ULDataReader.
GetString method (inherited from ULDataReader)	Returns the value for the specified column as a String.
GetTimeSpan method (inherited from ULDataReader)	Returns the value for the specified column as a TimeSpan with millisecond accuracy.
GetUInt16 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt16.
GetUInt32 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt32.
GetUInt64 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt64.
GetValue method (inherited from ULDataReader)	Returns the value of the specified column in its native format.
GetValues method (inherited from ULDataReader)	Returns all the column values for the current row.
Insert method	Inserts a new row with the current column values (specified using the set methods). Each insert must be preceded by a call to "InsertBegin method" on page 844.
InsertBegin method	Prepares to insert a new row into the table by setting all current column values to their default values.

Member name	Description
IsDBNull method (inherited from ULDataReader)	Checks whether the value from the specified column is NULL.
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.
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.
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.
MoveAfterLast method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to after the last row of the cursor.
MoveBeforeFirst method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to before the first row of the cursor.
MoveFirst method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the first row of the cursor.
MoveLast method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the last row of the cursor.
MoveNext method (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 (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the previous row or before the first row.
MoveRelative method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor relative to the current row.
NextResult method (inherited from ULDataReader)	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
Read method (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 (inherited from ULResultSet)	Sets the value for the specified column using a Boolean.
SetByte method (inherited from ULResultSet)	Sets the value for the specified column using a Byte (unsigned 8-bit integer).
SetBytes method (inherited from ULResultSet)	Sets the value for the specified column using an array of Bytes.
SetDBNull method (inherited from ULResultSet)	Sets a column to NULL.

Member name	Description
SetDateTime method (inherited from ULResultSet)	Sets the value for the specified column using a DateTime.
SetDecimal method (inherited from ULResultSet)	Sets the value for the specified column using a Decimal.
SetDouble method (inherited from ULResultSet)	Sets the value for the specified column using a Double.
SetFloat method (inherited from ULResultSet)	Sets the value for the specified column using a Single.
SetGuid method (inherited from ULResultSet)	Sets the value for the specified column using a Guid.
SetInt16 method (inherited from ULResultSet)	Sets the value for the specified column using an Int16.
SetInt32 method (inherited from ULResultSet)	Sets the value for the specified column using an Int32.
SetInt64 method (inherited from ULResultSet)	Sets the value for the specified column using an Int64.
SetString method (inherited from ULResultSet)	Sets the value for the specified column using a String.
SetTimeSpan method (inherited from ULResultSet)	Sets the value for the specified column using a TimeSpan.
SetToDefault method (inherited from ULResultSet)	Sets the value for the specified column to its default value.
SetUInt16 method (inherited from ULResultSet)	Sets the value for the specified column using a UInt16.
SetUInt32 method (inherited from ULResultSet)	Sets the value for the specified column using an UInt32.
SetUInt64 method (inherited from ULResultSet)	Sets the value for the specified column using a UInt64.
Truncate method	Deletes all rows in the table while temporarily activating a stop synchronization delete.
Update method (inherited from ULResultSet)	Updates the current row with the current column values (specified using the set methods).
UpdateBegin method	Prepares to update the current row in the table.

See also

- "ULTable class" on page 830

Schema property

Holds the table schema. This property is only valid while its connection is open.

Prototypes

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

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

Property value

The "ULTableSchema class" on page 849 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 "GetSchemaTable method" on page 624.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830

DeleteAllRows method

Deletes all rows in the table.

Prototypes

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 "StopSynchronizationDelete method" on page 526.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "Truncate method" on page 848

FindBegin method

Prepares to perform a new Find on a table.

Prototypes

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 830
- "ULTable members" on page 830
- "FindFirst() method" on page 837
- "FindFirst(Int16) method" on page 838
- "FindLast() method" on page 839
- "FindLast(Int16) method" on page 839

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.

Prototypes

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 ("IsEOF property" on page 607).

Each search must be preceded by a call to "FindBegin method" on page 836.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "FindFirst methods" on page 837
- "FindBegin method" on page 836
- "FindNext() method" on page 840
- "FindPrevious() method" on page 842
- "FindFirst(Int16) method" on page 838

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.

Prototypes

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 ("IsEOF property" on page 607).

Each search must be preceded by a call to "FindBegin method" on page 836.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "FindFirst methods" on page 837
- "FindBegin method" on page 836
- "FindNext(Int16) method" on page 841
- "FindPrevious(Int16) method" on page 842
- "FindFirst() method" on page 837

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.

Prototypes

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 ("IsBOF property" on page 607).

Each search must be preceded by a call to "FindBegin method" on page 836.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "FindLast methods" on page 839
- "FindBegin method" on page 836
- "FindNext() method" on page 840
- "FindPrevious() method" on page 842
- "FindLast(Int16) method" on page 839

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.

Prototypes

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 ("IsBOF property" on page 607).

Each search must be preceded by a call to "FindBegin method" on page 836.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "FindLast methods" on page 839
- "FindBegin method" on page 836
- "FindNext(Int16) method" on page 841
- "FindPrevious(Int16) method" on page 842
- "FindLast() method" on page 839

FindNext methods

Continues a "FindFirst() method" on page 837 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 "FindFirst() method" on page 837 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.

Prototypes

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 ("IsEOF property" on page 607).

FindNext behavior is undefined if the column values being searched for are modified during a row update.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "FindNext methods" on page 840
- "FindFirst() method" on page 837
- "FindNext(Int16) method" on page 841

FindNext(Int16) method

Continues a "FindFirst() method" on page 837 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.

Prototypes

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 ("IsEOF property" on page 607).

FindNext behavior is undefined if the column values being searched for are modified during a row update.

See also

- "ULTable class" on page 830

- "ULTable members" on page 830
- "FindNext methods" on page 840
- "FindFirst(Int16) method" on page 838
- "FindNext() method" on page 840

FindPrevious methods

Continues a "FindLast() method" on page 839 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 "FindLast() method" on page 839 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.

Prototypes

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 ("IsBOF property" on page 607).

FindPrevious behavior is undefined if the column values being searched for are modified during a row update.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "FindPrevious methods" on page 842
- "FindLast() method" on page 839
- "FindPrevious(Int16) method" on page 842

FindPrevious(Int16) method

Continues a "FindLast() method" on page 839 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.

Prototypes

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 ("IsBOF property" on page 607).

FindPrevious behavior is undefined if the column values being searched for are modified during a row update.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "FindPrevious methods" on page 842
- "FindLast(Int16) method" on page 839
- "FindPrevious() method" on page 842

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 "InsertBegin method" on page 844.

Prototypes

Visual Basic
```
Public Sub Insert()
```

C#
```
public void Insert();
```

See also

- "ULTable class" on page 830
- "ULTable members" on page 830

InsertBegin method

Prepares to insert a new row into the table by setting all current column values to their default values.

Prototypes

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" on page 843, and that change is not made permanent until it is committed.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "Insert method" on page 843

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.

Prototypes

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 ("IsBOF property" on page 607).

Each search must be preceded by a call to "LookupBegin method" on page 846.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "LookupBackward methods" on page 844
- "LookupBegin method" on page 846
- "LookupBackward(Int16) method" on page 845

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.

Prototypes

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 ("IsBOF property" on page 607).

Each search must be preceded by a call to "LookupBegin method" on page 846.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "LookupBackward methods" on page 844
- "LookupBegin method" on page 846
- "LookupBackward() method" on page 844

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.

Prototypes

Visual Basic
Public Sub **LookupBegin()**

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

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "LookupForward() method" on page 846
- "LookupForward(Int16) method" on page 847
- "LookupBackward() method" on page 844
- "LookupBackward(Int16) method" on page 845

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.

Prototypes

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 ("IsEOF property" on page 607).

Each search must be preceded by a call to "LookupBegin method" on page 846.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "LookupForward methods" on page 846
- "LookupBegin method" on page 846
- "LookupForward(Int16) method" on page 847

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.

Prototypes

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 ("IsEOF property" on page 607).

Each search must be preceded by a call to "LookupBegin method" on page 846.

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "LookupForward methods" on page 846
- "LookupBegin method" on page 846
- "LookupForward() method" on page 846

Truncate method

Deletes all rows in the table while temporarily activating a stop synchronization delete.

Prototypes

Visual Basic
Public Sub **Truncate()**

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

See also

- "ULTable class" on page 830
- "ULTable members" on page 830
- "DeleteAllRows method" on page 836

UpdateBegin method

Prepares to update the current row in the table.

Prototypes

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 "Update method" on page 744, 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 830
- "ULTable members" on page 830

ULTableSchema class

UL Ext.: Represents the schema of an UltraLite table. This class cannot be inherited.

Prototypes

Visual Basic
Public NotInheritable Class **ULTableSchema**
  Inherits ULCursorSchema

C#
public sealed class **ULTableSchema** : ULCursorSchema

Remarks

There is no constructor for this class. A "ULTableSchema class" on page 849 object is attached to a table as its "Schema property" on page 836.

Inherits: "ULCursorSchema class" on page 566

See also

- "ULTableSchema members" on page 849

ULTableSchema members

Public properties

Member name	Description
ColumnCount property (inherited from ULCursorSchema)	Returns the number of columns in the cursor.
IndexCount property	Returns the number of indexes on the table.
IsNeverSynchronized property	Checks whether the table is marked as never being synchronized.
IsOpen property (inherited from ULCursorSchema)	Checks whether the cursor schema is currently open.
Name property	Returns the name of the table.
PrimaryKey property	Returns the index schema of the primary key for the table.
UploadUnchangedRows property	Checks whether the database uploads rows that have not changed.

Public methods

Member name	Description
GetColumnDefaultValue method	Returns the default value of the specified column.

Member name	Description
GetColumnID method (inherited from ULCursorSchema)	Returns the column ID of the named column.
GetColumnName method (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
GetColumnPartitionSize method	Returns the global autoincrement partition size assigned to the specified column.
GetColumnPrecision method (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 (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
GetColumnScale method (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 (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 (inherited from ULCursorSchema)	Returns the UltraLite.NET data type of the column identified by the specified column ID.
GetIndex method	Returns the index schema of the named index.
GetIndexName method	Returns the name of the index identified by the specified index ID.
GetOptimalIndex method	The optimal index for searching a table using the specified column.
GetPublicationPredicate method	Returns the publication predicate for this table in the named publication.
GetSchemaTable method (inherited from ULCursorSchema)	Returns a DataTable that describes the column schema of the “ULDataReader class” on page 602.
IsColumnAutoIncrement method	Checks whether the specified column's default is set to autoincrement.
IsColumnCurrentDate method	Checks whether the specified column's default is set to the current date (“ULDbType enumeration” on page 637).
IsColumnCurrentTime method	Checks whether the specified column's default is set to the current time (“ULDbType enumeration” on page 637).
IsColumnCurrentTimestamp method	Checks whether the specified column's default is set to the current timestamp (“ULDbType enumeration” on page 637).
IsColumnGlobalAutoIncrement method	Checks whether the specified column's default is set to global autoincrement.

Member name	Description
IsColumnNewUUID method	Checks whether the specified column's default is set to a new UUID (Guid).
IsColumnNullable method	Checks whether the specified column is nullable.
IsInPublication method	Checks whether the table is contained in the named publication.

See also

- "ULTableSchema class" on page 849

IndexCount property

Returns the number of indexes on the table.

Prototypes

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 849
- "ULTableSchema members" on page 849

IsNeverSynchronized property

Checks whether the table is marked as never being synchronized.

Prototypes

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 849
- "ULTableSchema members" on page 849

Name property

Returns the name of the table.

Prototypes

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 849
- "ULTableSchema members" on page 849

PrimaryKey property

Returns the index schema of the primary key for the table.

Prototypes

Visual Basic
Public Readonly Property **PrimaryKey** As ULIndexSchema

C#
public ULIndexSchema **PrimaryKey** { get;}

Property value

A "ULIndexSchema class" on page 667 object representing the primary key for the table.

See also

- "ULTableSchema class" on page 849
- "ULTableSchema members" on page 849

UploadUnchangedRows property

Checks whether the database uploads rows that have not changed.

Prototypes

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 849
- "ULTableSchema members" on page 849

GetColumnDefaultValue method

Returns the default value of the specified column.

Prototypes

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,"ColumnCount property" on page 567-1]. 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 849

- "ULTableSchema members" on page 849

GetColumnPartitionSize method

Returns the global autoincrement partition size assigned to the specified column.

Prototypes

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,"ColumnCount property" on page 567-1]. The first column in the table has an ID value of zero.

Return value

The column's global autoincrement partition size as a UInt64.

Remarks

All global autoincrement columns in a given table share the same global autoincrement partition.

See also

- "ULTableSchema class" on page 849
- "ULTableSchema members" on page 849
- "IsColumnGlobalAutoIncrement method" on page 859

GetIndex method

Returns the index schema of the named index.

Prototypes

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 class" on page 667 object representing the named index.

See also

- "ULTableSchema class" on page 849
- "ULTableSchema members" on page 849

GetIndexName method

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

Prototypes

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 property" on page 851].

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 849
- "ULTableSchema members" on page 849
- "IndexCount property" on page 851

GetOptimalIndex method

The optimal index for searching a table using the specified column.

Prototypes

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,"ColumnCount property" on page 567-1]. The first column in the table has an ID value of zero.

Return value

A "ULIndexSchema class" on page 667 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 849
- "ULTableSchema members" on page 849

GetPublicationPredicate method

Returns the publication predicate for this table in the named publication.

Prototypes

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 849

- "ULTableSchema members" on page 849

IsColumnAutoIncrement method

Checks whether the specified column's default is set to autoincrement.

Prototypes

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,"ColumnCount property" on page 567-1]. 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 849
- "ULTableSchema members" on page 849

IsColumnCurrentDate method

Checks whether the specified column's default is set to the current date ("ULDbType enumeration" on page 637).

Prototypes

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,"ColumnCount property" on page 567-1]. 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 849
- "ULTableSchema members" on page 849

IsColumnCurrentTime method

Checks whether the specified column's default is set to the current time ("ULDbType enumeration" on page 637).

Prototypes

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,"ColumnCount property" on page 567-1]. 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 849
- "ULTableSchema members" on page 849

IsColumnCurrentTimestamp method

Checks whether the specified column's default is set to the current timestamp ("ULDbType enumeration" on page 637).

Prototypes

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,"ColumnCount property" on page 567-1]. 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 849
- "ULTableSchema members" on page 849

IsColumnGlobalAutoIncrement method

Checks whether the specified column's default is set to global autoincrement.

Prototypes

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,"ColumnCount property" on page 567-1]. 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 849
- "ULTableSchema members" on page 849
- "GetColumnPartitionSize method" on page 854
- "DatabaseID property" on page 505

IsColumnNewUUID method

Checks whether the specified column's default is set to a new UUID (Guid).

Prototypes

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,"ColumnCount property" on page 567-1]. 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 849
- "ULTableSchema members" on page 849

IsColumnNullable method

Checks whether the specified column is nullable.

Prototypes

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,"ColumnCount property" on page 567-1]. 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 849
- "ULTableSchema members" on page 849

IsInPublication method

Checks whether the table is contained in the named publication.

Prototypes

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 849
- "ULTableSchema members" on page 849

ULTransaction class

Represents a SQL transaction. This class cannot be inherited.

Prototypes

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 "BeginTransaction() method" on page 510. To associate a command with a transaction, use the "Transaction property" on page 472.

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: DbTransaction

Implements: IDbTransaction, IDisposable

See also

- "ULTransaction members" on page 862

ULTransaction members

Public properties

Member name	Description
Connection property	Returns the connection associated with the transaction.
IsolationLevel property	Returns the isolation level for the transaction.

Public methods

Member name	Description
Commit method	Commits the database transaction.
Dispose (inherited from Db-Transaction)	Releases the unmanaged resources used by the DbTransaction.
Rollback method	Rolls back the transaction's outstanding changes to the database.

See also

- "ULTransaction class" on page 862

Connection property

Returns the connection associated with the transaction.

Prototypes

Visual Basic
Public Readonly Property **Connection** As ULConnection

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

Property value

The "ULConnection class" on page 498 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 IDbTransaction.Connection and DbCommand.Connection.

See also

- "ULTransaction class" on page 862
- "ULTransaction members" on page 862
- "BeginTransaction() method" on page 510

IsolationLevel property

Returns the isolation level for the transaction.

Prototypes

Visual Basic
Public Overrides Readonly Property **IsolationLevel** As IsolationLevel

C#
public override IsolationLevel **IsolationLevel** { get;}

Property value

One of the IsolationLevel values. UltraLite.NET only supports IsolationLevel.ReadUncommitted.

See also

- "ULTransaction class" on page 862
- "ULTransaction members" on page 862
- "BeginTransaction() method" on page 510

Commit method

Commits the database transaction.

Prototypes

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.

See also

- "ULTransaction class" on page 862
- "ULTransaction members" on page 862
- "Rollback method" on page 864

Rollback method

Rolls back the transaction's outstanding changes to the database.

Prototypes

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 862
- "ULTransaction members" on page 862
- "Commit method" on page 864

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

