Contents

About this book ........................................................................................................ vii

System requirements and supported platforms ................................................. 1

UltraLite.NET application development .......................................................... 3
  SQL Anywhere tools in Visual Studio ................................................................. 3
  Connection setup for an UltraLite database ...................................................... 3
  Data creation and modification using SQL statements ................................... 5
  Data creation and modification using the ULTable class ............................... 9
  Transaction management ................................................................................. 15
  Schema information access ............................................................................. 16
  Error handling .................................................................................................. 17
  MobiLink data synchronization ..................................................................... 18
  How to build and deploy UltraLite.NET applications .................................. 19

Tutorial: Building a Windows Mobile application using UltraLite.NET .......... 25
  Lesson 1: Creating a Visual Studio project ...................................................... 26
  Lesson 2: Creating an UltraLite database ....................................................... 29
  Lesson 3: Adding database connection controls to the application .......... 30
  Lesson 4: Inserting, updating, and deleting data ......................................... 32
  Lesson 5: Building and deploying the application .................................... 37
  Code listing for C# tutorial ............................................................................ 38
  Code listing for Visual Basic tutorial ............................................................. 40

UltraLite.NET API reference ................................................................................. 43
  ULAbsenceSyncListener interface ................................................................. 44
  ULBulkCopy class ......................................................................................... 46
  ULBulkCopyColumnMapping class ............................................................... 56
  ULBulkCopyColumnMappingCollection class ............................................ 62
  ULCommand class ......................................................................................... 71
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULCommandBuilder class</td>
<td>107</td>
</tr>
<tr>
<td>ULConnection class</td>
<td>118</td>
</tr>
<tr>
<td>ULConnectionParms class</td>
<td>163</td>
</tr>
<tr>
<td>ULConnectionStringBuilder class</td>
<td>172</td>
</tr>
<tr>
<td>ULCreateParms class</td>
<td>188</td>
</tr>
<tr>
<td>ULCursorSchema class</td>
<td>197</td>
</tr>
<tr>
<td>ULDataAdapter class</td>
<td>204</td>
</tr>
<tr>
<td>ULDatabaseManager class</td>
<td>214</td>
</tr>
<tr>
<td>ULDatabaseSchema class</td>
<td>221</td>
</tr>
<tr>
<td>ULDataReader class</td>
<td>228</td>
</tr>
<tr>
<td>ULException class</td>
<td>265</td>
</tr>
<tr>
<td>ULFactory class</td>
<td>267</td>
</tr>
<tr>
<td>ULFileTransfer class</td>
<td>272</td>
</tr>
<tr>
<td>ULFileTransferProgressData class</td>
<td>287</td>
</tr>
<tr>
<td>ULFileTransferProgressListener interface</td>
<td>289</td>
</tr>
<tr>
<td>ULIndexSchema class</td>
<td>290</td>
</tr>
<tr>
<td>ULInfoMessageEventArgs class</td>
<td>297</td>
</tr>
<tr>
<td>ULMetaDataCollectionNames class</td>
<td>299</td>
</tr>
<tr>
<td>ULPParameter class</td>
<td>307</td>
</tr>
<tr>
<td>ULPParameterCollection class</td>
<td>321</td>
</tr>
<tr>
<td>ULResultSet class</td>
<td>339</td>
</tr>
<tr>
<td>ULResultSetSchema class</td>
<td>362</td>
</tr>
<tr>
<td>ULRowsCopiedEventArgs class</td>
<td>364</td>
</tr>
<tr>
<td>ULRowUpdatedEventArgs class</td>
<td>366</td>
</tr>
<tr>
<td>ULRowUpdatingEventArgs class</td>
<td>369</td>
</tr>
<tr>
<td>ULServerSyncListener interface</td>
<td>371</td>
</tr>
<tr>
<td>ULSqlProgressData class</td>
<td>373</td>
</tr>
<tr>
<td>ULSyncParms class</td>
<td>375</td>
</tr>
<tr>
<td>ULSyncProgressData class</td>
<td>386</td>
</tr>
<tr>
<td>ULSyncProgressListener interface</td>
<td>396</td>
</tr>
<tr>
<td>ULSyncResult class</td>
<td>397</td>
</tr>
<tr>
<td>ULTuple class</td>
<td>401</td>
</tr>
<tr>
<td>ULTupleSchema class</td>
<td>421</td>
</tr>
<tr>
<td>ULTransaction class</td>
<td>434</td>
</tr>
<tr>
<td>ULIInfoMessageEventHandler delegate</td>
<td>437</td>
</tr>
</tbody>
</table>
ULRowUpdatedEventHandler delegate ................................................................. 438
ULRowUpdatingEventHandler delegate ............................................................... 438
ULRowsCopiedEventHandler delegate ................................................................. 439
ULSyncProgressedDlg delegate ........................................................................... 439
ULAuthStatusCode enumeration .......................................................................... 440
ULBulkCopyOptions enumeration ....................................................................... 441
ULDBValid enumeration ....................................................................................... 441
ULDateOrder enumeration .................................................................................... 442
ULDbType enumeration .......................................................................................... 442
ULRuntimeType enumeration ................................................................................. 446
ULSqlProgressState enumeration .......................................................................... 447
ULStreamType enumeration .................................................................................... 448
ULSyncProgressState enumeration ........................................................................ 449

Index .................................................................................................................... 451
About this book

This book describes the UltraLite.NET programming interface. With UltraLite.NET you can develop and deploy database applications to Windows computers, or handheld, mobile, or embedded devices.
System requirements and supported platforms

Development platforms
To develop applications using UltraLite.NET, you must have the following:

● A supported desktop version of Microsoft Windows.
● For Windows Mobile devices, .NET Compact Framework version 2 or later.

Target platforms
UltraLite.NET supports the following target platforms:

● Microsoft .NET Compact Framework version 2.0 or later and .NET Framework 2.0 or later on Windows.

● For Windows Mobile devices, Microsoft .NET Compact Framework version 2 or later.

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

The UltraLite.NET API provides the iAnywhere.Data.UltraLite namespace. 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.

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

**Note**
You can use the UltraLite C/C++ API as an alternative to the UltraLite.NET API to create applications for Windows Mobile devices and desktop. For more information, see “UltraLite C++ application development” [UltraLite - C and C++ Programming].

SQL Anywhere tools in Visual Studio

Visual Studio integration is supported for UltraLite.NET. You can access the SQL Anywhere integration tools from the Visual Studio Server Explorer in Visual Studio 2005 or later.

Connection setup for an UltraLite database

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

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

Using the 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.

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.

- **User authentication**  
  You can change the user ID and password for the application from the default values of DBA and sql, respectively, by using methods. Each UltraLite database can define a maximum of four user IDs.
Synchronization
A set of objects governing synchronization is accessed from the Connection object.

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

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

Multi-threaded applications
Each ULConnection object 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.

See also
- “Transaction management” on page 15
- “MobiLink data synchronization” on page 18
- “Data creation and modification using the ULTable class” on page 9
- “Data creation and modification using SQL statements” on page 5
- “ULConnection class [UltraLite.NET]” on page 118

Connecting to an UltraLite database
Use the ULConnectionParms object to connect to an UltraLite database named mydata.udb.

Prerequisites
There are no prerequisites for this task.

Task
1. Declare a ULConnection object.

   ULConnection conn;

2. Open a connection to an existing database.

   You can specify connection parameters either as a connection string or using the ULConnectionParms object.

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

Results
A connection to the mydata.udb database is established.
Use the database connection to perform SQL operations that create, modify, or delete data. You can modify existing database options on an open connection. Close the ULConnection object when finished.

Data creation and modification using SQL statements

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

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.

See also

- “Data creation and modification using the ULTable class” on page 9
- “SQL statements” [SQL Anywhere Server - SQL Reference]

Data modification using INSERT, UPDATE, and DELETE

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

See also

- “ULCommand.ExecuteNonQuery method [UltraLite.NET]” on page 91

Inserting a row in a table

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.

Prerequisites

There are no prerequisites for this task.
Task

1. Declare a ULCommand.

   ULCommand cmd;

2. Assign a SQL statement to the ULCommand object.

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

3. Assign input parameter values for the statement.

   The following code shows a string parameter.
   
   String newValue;
   cmd.Parameters.Clear();
   // assign value
   cmd.Parameters.Add("", newValue);

4. Execute the statement.

   The return value indicates the number of rows affected by the statement.
   
   int rowsInserted = cmd.ExecuteNonQuery();

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

   myTransaction.Commit();

Results

A new row is added to MyTable where the MyColumn value is set to an empty string.

Updating a row in a table

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.

Prerequisites

There are no prerequisites for this task.

Task

1. Declare a ULCommand.

   ULCommand cmd;

2. Assign a statement to the ULCommand object.
cmd = conn.CreateCommand();
cmd.CommandText = "UPDATE MyTable SET MyColumn1 = ? WHERE MyColumn2 = ?";

3. Assign input parameter values for the statement.

    String newValue;
    String oldValue;

    cmd.Parameters.Clear();

    // assign values
    cmd.Parameters.Add("", newValue);
    cmd.Parameters.Add("", oldValue);

4. Execute the statement.

    int rowsUpdated = cmd.ExecuteNonQuery();

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

    myTransaction.Commit();

Results

Row entries from MyTable are updated where the MyColumn1 value is an empty string. In this scenario, the MyColumn2 value is also set to an empty string.

Deleting a row in a table

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.

Prerequisites

There are no prerequisites for this task.

Task

1. Declare a ULCommand.

    ULCommand cmd;

2. Assign a statement to the ULCommand object.

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

3. Assign input parameter values for the statement.

    String deleteValue;
    cmd.Parameters.Clear();

    // assign value
    cmd.Parameters.Add("", deleteValue);
4. Execute the statement.

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

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

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

Results

Row entries from MyTable are deleted where the MyColumn value in the table is an empty string.

**Retrieving data using SELECT**

Execute a SELECT statement to retrieve information from an UltraLite database and handle the result set that is returned.

**Prerequisites**

There are no prerequisites for this task.

**Task**

1. Declare a ULCommand object to hold the query.

   ```csharp
   ULCommand cmd;
   ```

2. Assign a statement to the object.

   ```csharp
   cmd = conn.CreateCommand();
   cmd.CommandText = "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.

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

Results

The result of the SELECT statement contains a string, which is then output to the command prompt.
Result set schema description

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

Example

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

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

SQL result set navigation

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.

Data creation and modification using the ULTable class

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

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.

See also

- “Data creation and modification using SQL statements” on page 5

Row navigation

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

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

- **MoveAfterLast** moves to a position after the last row.
- **MoveBeforeFirst** moves to a position before the first row.
- **MoveFirst** moves to the first row.
- **MoveLast** moves to the last row.
- **MoveNext** moves to the next row.
- **MovePrevious** moves to the previous row.
- **MoveRelative(offset)** moves a certain number of rows relative to the current row, as specified by the offset. Positive offset values move forward in the table, relative to the current position of the cursor in the table, and negative offset values move backward in the table. An offset value of zero does not move the cursor, but allows you to repopulate the row buffer.

See also

- “ULTable class [UltraLite.NET]” on page 401
- “ULTableSchema class [UltraLite.NET]” on page 421

Example

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

```csharp
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.

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

**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.

**Row insertion**

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.

```csharp
   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.
Row updates

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

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

See also
●  “Transaction management” on page 15

Updating a row in a table

Use the Update method to update a row in a table.

Prerequisites

There are no prerequisites for this task.

Task

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 a table.

   ```
   t.UpdateBegin();
   ```

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();
   ```

Results

The current row is updated. If you changed the value of a column in the index specified when the Table object was opened, the current row is undefined.
Row searches

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.

See also
- “ULTable class [UltraLite.NET]” on page 401

Searching for a row with the find and lookup methods

Use the find and lookup methods to search for a row in a ULTable object.

Prerequisites

There are no prerequisites for this task.

Task

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.

   ```csharp
   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.

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

3. Search for the row.

   Use the appropriate method to perform 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.

```csharp
t.FindFirst();
```

4. Search for the next instance of the row.

Use the appropriate method to perform 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.

**Results**

The cursor points to the desired row.

**Next**

Perform operations on the row, such as delete, or modify data that pertains to the row.

### Row retrieval

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.

```csharp
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.

```csharp
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.

```csharp
    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.

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

Casting values

The method you choose must match the data type you want to assign. UltraLite automatically casts database data types where they are compatible, so that you could use the getString method to fetch an integer value into a string variable, and so on.

See also

- “CAST function [Data type conversion]” [UltraLite - Database Management and Reference]
- “CONVERT function [Data type conversion]” [UltraLite - Database Management and Reference]

Row deletions

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.

You delete a row by moving the cursor to the row you want to delete and then executing the Table.Delete method.

Example

The following code illustrates how to delete the first row in a table:

```csharp
    t.MoveFirst();
    t.Delete();
```

Transaction management

UltraLite provides transaction processing to ensure the integrity of the data in your database. A transaction is a logical unit of work. Either an entire transaction is executed, or none of the statements in the transaction are executed.

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

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

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

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

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

**Note**

UltraLite supports only the ReadCommitted and ReadUncommitted members of the IsolationLevel enumeration.

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

**See also**

- “ULConnection class [UltraLite.NET]” on page 118
- “ULTransaction class [UltraLite.NET]” on page 434
- “ULConnection.BeginTransaction method [UltraLite.NET]” on page 125

### Schema information access

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

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

You can access the following schema objects and information:

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

  Call ULConnection.Schema to obtain a ULDatabaseSchema object.

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

  Call ULTable.Schema to obtain a ULTableSchema object.

- **ULIndexSchema** Information about the column in the index. As an index has no data directly associated with it there is no separate Index class, just a ULIndexSchema class.
Call the ULTableSchema.GetIndex, ULTableSchema.GetOptimalIndex, or ULTableSchema.GetPrimaryKey method to obtain a ULIndexSchema object. See and .

See also

- “ULTable.Schema property [UltraLite.NET]” on page 421
- “ULDatabaseSchema class [UltraLite.NET]” on page 221
- “ULTableSchema class [UltraLite.NET]” on page 421
- “ULIndexSchema class [UltraLite.NET]” on page 290

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.

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

```csharp
public void Sync() {
    try {
        _conn.Synchronize( this );
        _inSync = false;
    }
    catch( ULException uEx ) {
        if( uEx.NativeError == ULSQLCode.SQLCODE_MOBILINK_COMMUNICATIONS_ERROR ) {
            MessageBox.Show(
                "StreamErrorParameters = " + _conn.SyncResult.StreamErrorParameters + "\r\n" + 
                "StreamErrorSystem = " + _conn.SyncResult.StreamErrorSystem + "\r\n"
            );
        } else {
            MessageBox.Show(uEx.Message);
        }
    }
    catch(System.Exception ex ) {
        MessageBox.Show(ex.Message);
    }
}
```

See also

- “Error Messages”
- “ULSyncProgressListener interface [UltraLite.NET]” on page 396
- “ULSyncResult class [UltraLite.NET]” on page 397
MobiLink data synchronization

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.

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 16 installation.

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

Note
Separately licensed component required.

FIPS-certified encryption requires a separate license. All strong encryption technologies are subject to export regulations.

See “Separately licensed components” [SQL Anywhere 16 - Introduction].

See also
● “UltraLite clients” [UltraLite - Database Management and Reference]

Synchronization initiation in a C# application

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

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

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

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

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

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

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

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.

See also

- “ULActiveSyncListener interface [UltraLite.NET]” on page 44

How to build and deploy UltraLite.NET applications

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

See also

- “UltraLite application build and deployment specifications” [UltraLite - Database Management and Reference]
- “Deploying the ActiveSync provider for UltraLite” [UltraLite - Database Management and Reference]
Deploying an UltraLite.NET application for Windows Mobile

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, references, and deployment files to ensure that your UltraLite.NET application runs successfully on Windows and Windows Mobile devices.

Prerequisites

There are no prerequisites for this task.

Task

1. Specify the following parameters:
   
   ● When using obfuscation, set the creation parameter `obfuscate=1` while creating the database.
   
   ● When using AES, set the connection parameter `DBKEY=encryption-key` while creating or connecting to the database.

2. Set the appropriate parameter settings when using synchronization in your UltraLite application:

<table>
<thead>
<tr>
<th>Synchronization type</th>
<th>Parameter settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP</td>
<td>Set the <strong>Stream</strong> synchronization parameter to <code>tcpip</code>.</td>
</tr>
<tr>
<td>HTTP</td>
<td>Set the <strong>Stream</strong> synchronization parameter to <code>http</code>.</td>
</tr>
<tr>
<td>RSA TLS</td>
<td>Set the <strong>Stream</strong> synchronization parameter to <code>tls</code>.</td>
</tr>
<tr>
<td>RSA HTTPS</td>
<td>Set the <strong>Stream</strong> synchronization parameter to <code>https</code>.</td>
</tr>
</tbody>
</table>

3. When using RSA end-to-end encryption, set the protocol option `e2ee_public_key=key-file`.

4. When using ZLIB compression, set the protocol option `compression=zlib`.

5. Add references to:

   ● `iAnywhere.Data.UltraLite`
   
   ● `iAnywhere.Data.UltraLite.resources`

6. Deploy the following files:

   ● `%SQLANY16%UltraLite\UltraLite.NET\Assembly\V2\iAnywhere.Data.UltraLite.dll`
   
   ● `%SQLANY16%UltraLite\UltraLite.NET\Assembly\V2\en\iAnywhere.Data.UltraLite.resources.dll`
7. Deploy the files appropriate for your application:

- When using ZLIB compression, `mlczlib16.dll`.
- When using RSA TLS, RSA HTTPS, or RSA E2EE, `mlcrsa16.dll`.

For Windows Mobile, the files are located in `%SQLANY16%\UltraLite\CE\Arm50`. For Windows, the files are located in `%SQLANY16%\UltraLite\Windows\x64` or `%SQLANY16%\UltraLite\Windows\x86`.

**Results**

The UltraLite.NET application runs successfully on the Windows desktop or Windows Mobile device that it is deployed to.

**Next**

Deploy an UltraLite database to the Windows desktop or Windows Mobile device that the application was deployed to, or create a new database with the deployed application.

**See also**

- “UltraLite application build and deployment specifications” [UltraLite - Database Management and Reference]
- “UltraLite and UltraLite Java edition database deployment techniques” [UltraLite - Database Management and Reference]

---

**Deploying an UltraLite.NET application for Windows Mobile (UltraLite engine)**

[This topic has been updated for build 1823.]

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, references, and deployment files to ensure that your UltraLite.NET application runs successfully on Windows and Windows Mobile devices.

**Prerequisites**

There are no prerequisites for this task.

**Task**

1. Specify the following parameters:

   - When using obfuscation, set the creation parameter `obfuscate=1` while creating the database.
   - When using AES encryption, set the connection parameter `DBKEY=encryption-key` while creating or connecting to the database.
2. Set the appropriate parameter settings when using synchronization in your UltraLite application:

<table>
<thead>
<tr>
<th>Synchronization type</th>
<th>Parameter settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP</td>
<td>Set the Stream synchronization parameter to tcpip.</td>
</tr>
<tr>
<td>HTTP</td>
<td>Set the Stream synchronization parameter to http.</td>
</tr>
<tr>
<td>RSA TLS</td>
<td>Set the Stream synchronization parameter to tls.</td>
</tr>
<tr>
<td>RSA HTTPS</td>
<td>Set the Stream synchronization parameter to https.</td>
</tr>
</tbody>
</table>

3. When using RSA end-to-end encryption, set the protocol option e2ee_public_key=key-file.

4. When using ZLIB compression, set the protocol option compression=zlib.

5. Add references to:
   - iAnywhere.Data.UltraLite
   - iAnywhere.Data.UltraLite.resources

6. Deploy the following files:
   - %SQLANY16%\UltraLite\UltraLite.NET\Assembly\V2\iAnywhere.Data.UltraLite.dll.
   - %SQLANY16%\UltraLite\UltraLite.NET\Assembly\V2\en\iAnywhere.Data.UltraLite.resources.dll.
   - ulnetclient16.dll, located in %SQLANY16%\UltraLite\UltraLite.NET\CE\Arm50 for Windows Mobile. For Windows, it is located in %SQLANY16%\UltraLite\UltraLite.NET\x64 or %SQLANY16%\UltraLite\UltraLite.NET\win32.

7. Deploy the files appropriate for your application:
   - uleng16.exe.
   - When using ZLIB compression, mlczlib16.dll.
   - When using RSA TLS, RSA HTTPS, or RSA E2EE, mlcrsa16.dll.

For Windows Mobile, the files are located in %SQLANY16%\UltraLite\CE\Arm.50. For Windows, the files are located in %SQLANY16%\UltraLite\Windows\x64 or %SQLANY16%\UltraLite\Windows\x86.

Results

The UltraLite.NET application, which uses the UltraLite engine, runs successfully on the Windows desktop or Windows Mobile device that it is deployed to.

Next

Deploy an UltraLite database to the Windows desktop or Windows Mobile device that the application was deployed to, or create a new database with the deployed application.
See also

- “UltraLite application build and deployment specifications” [UltraLite - Database Management and Reference]
- “UltraLite and UltraLite Java edition database deployment techniques” [UltraLite - Database Management and Reference]
Tutorial: Building a Windows Mobile application using UltraLite.NET

This tutorial guides you through the process of building an UltraLite application for Windows Mobile using Microsoft Visual Studio. It uses the ADO.NET interface provided by the iAnywhere.Data.UltraLite namespace and runs on the .NET 3.5 Compact Framework.

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

Competencies and experience

This tutorial assumes the following:

- You are familiar with the C# programming language or the Visual Basic programming language.
- You know how to create an UltraLite database using the UltraLite plug-in for Sybase Central.
- You have Microsoft Visual Studio installed on your computer and you are familiar with using Visual Studio. This tutorial is tested using Visual Studio 2008 and may refer to Visual Studio actions or procedures that may be slightly different in other versions of Visual Studio.
- You have installed the Windows Mobile 5.0 SDK or later from Microsoft
- You have installed the .NET 3.5 Compact Framework to your mobile device

Goals

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

Installation note

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

- Ensure Visual Studio is not running.
- For Visual Studio 2005 or later, run installULNet.exe from the folder named %SQLANY16%\UltraLite \UltraLite.NET\Assembly\v2\*. This task may require administrator privileges.

See also

- “Creating an UltraLite database with the Create Database Wizard” [UltraLite - Database Management and Reference]
Lesson 1: Creating a Visual Studio project

In this lesson, you create and configure a new Visual Studio application. You can choose whether to use Visual Basic or C# as your programming language.

Prerequisites

This lesson assumes that you have installed the required software. See “Tutorial: Building a Windows Mobile application using UltraLite.NET” on page 25.

Context and remarks

This tutorial assumes that if you are designing a C# application, your files are in the directory C:\tutorial\uldotnet\CSApp and that if you are designing a Visual Basic application, your files are in the directory C:\tutorial\uldotnet\VBApp. If you choose to use a directory with a different name, use that directory throughout the tutorial.

Task

1. Create a Visual Studio project.

   ● In the Visual Studio File menu, click New » Project.

   ● The New Project window appears. In the left pane, expand either the Visual Basic folder or the Visual C# folder. Click Smart Device for the project type.

   In the right pane, click a Smart Device Project and name your project VBApp or CSApp, depending on whether you are using Visual Basic or C# for the programming language.

   ● Enter a Location of C:\tutorial\uldotnet and click OK.

   ● Click Windows Mobile 5.0 Pocket PC SDK as the target platform and .NET Compact Framework Version 3.5 as the target .NET Compact Framework version. Click OK.

2. Add references to your project.

   ● Add the iAnywhere.Data.UltraLite assembly and the associated resources to your project.

     a. From the Project menu, click Add Reference.

     b. Click iAnywhere.Data.UltraLite and iAnywhere.Data.UltraLite EN (for English) in the list of available references. Click OK to add them to the list of selected components.

     If your desired language is not English, click Browse and locate iAnywhere.Data.UltraLite xx in the UltraLite\UltraLite.NET\ce\Assembly\v2\xx subdirectory of your SQL Anywhere installation, where xx is a two-letter abbreviation for your desired language (for example, use en for English). Click iAnywhere.Data.UltraLite.resources.dll and click Open.

   ● 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, click Add Existing Item and browse to the UltraLite\UltraLite.NET\ce subdirectory of your SQL Anywhere installation.
b. In the **Objects of Type** list, click **Executable Files**.

c. Open the folder corresponding to the processor of the Windows Mobile device you are using. For Visual Studio 2005 and later, open the `arm.50` folder. Click `ulnet16.dll`; Click the arrow on the **Add** button and click **Add as Link**.

3. Create a form for your application.

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Design - name</th>
<th>Appearance - text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button</td>
<td>btnInsert</td>
<td>Insert</td>
</tr>
<tr>
<td>Button</td>
<td>btnUpdate</td>
<td>Update</td>
</tr>
<tr>
<td>Button</td>
<td>btnDelete</td>
<td>Delete</td>
</tr>
<tr>
<td>TextBox</td>
<td>txtName</td>
<td>(no text)</td>
</tr>
<tr>
<td>ListBox</td>
<td>lbNames</td>
<td>(no text)</td>
</tr>
<tr>
<td>Label</td>
<td>laName</td>
<td>Name</td>
</tr>
</tbody>
</table>

Your form should look like the following figure:
4. Build and deploy your solution.

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

a. From the **Build** menu, click **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, click **Start Debugging**.

This action deploys your application to the mobile device or emulator, and starts it. The application is deployed to the emulator or device location: `\Program Files\VBA\App` or `\Program Files\CS\App` depending on your project name.

The deployment may take some time.

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

Results

The UltraLite.NET API is functional in the new Windows Mobile application.

Next

Proceed to “Lesson 2: Creating an UltraLite database” on page 29.

Lesson 2: Creating an UltraLite database

In this lesson, you create an UltraLite database using Sybase Central on a desktop PC.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Creating a Visual Studio project” on page 26.

Task

1. Click Start » Programs » SQL Anywhere 16 » Administration Tools » Sybase Central.

2. Use the UltraLite plug-in for Sybase Central to create a database in the same directory as your application.

   From the Tools menu, click UltraLite 16 » Create Database.

   In general, the default database characteristics provided by Sybase Central are suitable. Note the following characteristics:
   - Database file name: c:\tutorial\uldotnet\VBApp\VBApp.udb or c:\tutorial\uldotnet\CSApp\CSApp.udb, depending on your application type.
   - DBA user ID and password: Set to DBA and sql, respectively, for the purposes of examples in this documentation.
   - Collation sequence: Use the default collation.
   - Use case-sensitive string comparisons: This option should not be on.

   Click Finish and connect to the UltraLite database.

3. Create a new UltraLite table by highlighting the Tables folder icon in the Sybase Central tree view and then click File » New » Table. Note the following characteristics:
   - Table name: Type Names.
   - Columns: Create columns in the Names table with the following attributes:
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type (Size)</th>
<th>Nulls</th>
<th>Unique</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Integer</td>
<td>No</td>
<td>Yes (primary key)</td>
<td>Global autoincrement</td>
</tr>
<tr>
<td>Name</td>
<td>Varchar(30)</td>
<td>No</td>
<td>No</td>
<td>None</td>
</tr>
</tbody>
</table>

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

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

5. Link the initialized (empty) database file to your Visual Studio project so that the database file is deployed to the device along with the application code:
   - From the Visual Studio menu, click Project » Add Existing Item.
   - Ensure that Objects of Type is set to All Files. Browse to the directory where you created the database file and click the file VBAApp.udb or CSApp.udb depending on your application type.
   - Click the arrow in the Add button and click Add As Link.
   - In the Solution Explorer frame, right click the database file name that has just been added to the project and click Properties.
     In the properties panel, set the Build Action property to Content; set the Copy to Output Directory property to Copy always.

Results

An UltraLite database is created.

Next

Proceed to “Lesson 3: Adding database connection controls to the application” on page 30.

See also

- “Creating an UltraLite database with the Create Database Wizard” [UltraLite - Database Management and Reference]

**Lesson 3: Adding database connection controls to the application**

In this lesson, you add a control to your UltraLite.NET application that establishes a connection to an UltraLite database.

**Prerequisites**

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Creating a Visual Studio project” on page 26.
Task

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

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

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

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

   'Visual Basic
   Imports iAnywhere.Data.UltraLite

3. Add global variables to the form declaration.

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

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

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

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

   These variables are used as follows:

   • ULConnection A Connection object is the root object for all actions executed on a connection to a database.

   • ids The ids array is used to hold the ID column values returned after executing a query.

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

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

   This method performs the following tasks:

   • Open a connection to the database using the connection parameters set in the ulConnectionParms1 control.

   • Call the RefreshListBox method (defined later in this tutorial).

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

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

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

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

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

5. Build the project.

From the Build menu, click 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 current context. because RefreshListBox is not yet declared. The next lesson adds that function.

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

Results

The application is set up to connect to an UltraLite database.

Next

Proceed to “Lesson 4: Inserting, updating, and deleting data” on page 32.

Lesson 4: Inserting, updating, and deleting data

In this lesson, you add code to your application that uses Dynamic SQL to modify the data in your database.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Creating a Visual Studio project” on page 26.
Context and remarks

In this lesson, you create a supporting method to maintain the listbox. This approach is required for the data manipulation methods used in the remaining procedures.

Task

1. Right-click the form and click 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 C#, add the following code to your application as a method of the Form1 class.

```csharp
//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.MoveNext()
                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.
3. Build the project.

Building the project should result in no errors.

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

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

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

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

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

```vbnet
'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

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

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

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

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

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

    'Visual Basic
    Try
        Dim RowsUpdated As Long
        Dim updateID As Integer = ids(lbNames.SelectedIndex)
        Dim cmd As ULCommand = Conn.CreateCommand()
        cmd.CommandText = "UPDATE Names SET name = ? WHERE id = ?"
        cmd.Parameters.Add("", txtName.Text)
        cmd.Parameters.Add("", updateID)
        RowsUpdated = cmd.ExecuteNonQuery()
        cmd.Dispose()
        RefreshListBox()
    Catch
        MsgBox("Exception: " + Err.Description)
    End Try
6. On the form design tab, double-click **Delete** to create a btnDelete_Click method. Add code to perform 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 C#, add the following code to the btnDelete_Click method.

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

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

```vbnet
'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
```

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

Results

The Windows Mobile application is set up to perform data operations on the UltraLite database.

Next

Proceed to “Lesson 5: Building and deploying the application” on page 37.
Lesson 5: Building and deploying the application

In this lesson, you build your application and deploy it to a remote device or emulator.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Creating a Visual Studio project” on page 26.

Task

1. Build the solution.
   Ensure that your application builds without errors.

2. Choose the deployment target.
   The deployment target must match the version of ulnet16.dll that you included in your application.

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

4. 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.
   - Confirm that you chose Link File when adding the database file to the project and you set the Build Action to Content Only and Copy to Output Directory is set to Copy Always. If you did not set these options correctly, the files will not be deployed to the device.
   - Ensure that you added a reference to the correct version of ulnet16.dll for your target platform, or ran the Windows Mobile installer. For versions of Windows Mobile earlier than Windows Mobile 5.0, if you switch between the emulator and a real device, you must change the version of the library that you use.
   - 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.

5. Test your application:
a. Insert data into the database.
   Enter a name in the text box and click **Insert**. The name should now appear in the listbox.

b. Update data in the database.
   Click a name in 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.

c. Delete data from the database.
   Click a name in the list. Click **Delete**. The name no longer appears in the list.

**Results**

The application is tested and can be deployed.

**Code listing for C# tutorial**

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

```csharp
class Program
{
    static void Main()
    {
        Form1 form = new Form1();
        form.Show();
    }
}
```

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

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

        private void Form1_Load(object sender, EventArgs e)
        {
            try
            {
                string ConnString = "dbf=\Program Files\CSApp\CSApp.udb";
                Conn = new ULConnection(ConnString);
                Conn.Open();
                Conn.DatabaseID = 1;
                RefreshListBox();
            }
            catch (System.Exception t)
            {
                MessageBox.Show("Exception: " + t.Message);
            }
        }
        private void RefreshListBox()
        {
            try
            {
```
{ long NumRows;
    int I = 0;
    lbNames.Items.Clear();
    using (ULCommand cmd = Conn.CreateCommand())
    {
        cmd.CommandText = "SELECT ID, Name FROM Names";
        using (ULDataReader dr = cmd.ExecuteReader())
        {
            dr.MoveBeforeFirst();
            NumRows = dr.RowCount;
            ids = new int[NumRows];
            while (dr.MoveNext())
            {
                lbNames.Items.Add(
                    dr.GetString(1));
                ids[i] = dr.GetInt32(0);
                I++;
            }
            txtName.Text = " ";
        }
    }
    catch (Exception err)
    {
        MessageBox.Show("Exception in RefreshListBox: " + err.Message);
    }
}

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

private void btnUpdate_Click(object sender, EventArgs e)
{
    try
    {
        long RowsUpdated;
        int updateID = ids[lbNames.SelectedIndex];
        using (ULCommand cmd = Conn.CreateCommand())
        {
            cmd.CommandText = "UPDATE Names SET name = ? WHERE id = ?";
            cmd.Parameters.Add("", txtName.Text);
            cmd.Parameters.Add("", updateID);
            RowsUpdated = cmd.ExecuteNonQuery();
        }
    }
Imports iAnywhere.Data.UltraLite
Public Class Form1
    Dim Conn As ULConnection
    Dim ids() As Integer
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Try
            Dim ConnString As String = "dbf=\Program Files\VBAp\VBAp.udb"
            Conn = New ULConnection(ConnString)
            Conn.Open()
            Conn.DatabaseID = 1
            RefreshListBox()
        Catch
            MsgBox("Exception: " + Err.Description)
        End Try
    End Sub
    Private Sub RefreshListBox()
        Try
            Dim cmd As ULCommand = Conn.CreateCommand()
            Dim I As Integer = 0
            lbNames.Items.Clear()
            cmd.CommandText = "SELECT ID, Name FROM Names"
            Dim dr As ULDataReader = cmd.ExecuteReader()
            ReDim ids(dr.RowCount)
            While (dr.MoveNext)
                lbNames.Items.Add(dr.GetString(1))
                ids(I) = dr.GetInt32(0)
            End While
        Catch
            MsgBox("Exception: " + err.Message)
        End Try
    End Sub
    Private void btnDelete_Click(object sender, EventArgs e)
    { try
        long RowsDeleted;
        int deleteID = ids[lbNames.SelectedIndex];
        using (ULCommand cmd = Conn.CreateCommand())
        { cmd.CommandText = "DELETE From Names WHERE id = ?";
            cmd.Parameters.Add("", deleteID);
            RowsDeleted = cmd.ExecuteNonQuery();
        }
        RefreshListBox();
    } catch (Exception err)
    { MessageBox.Show("Exception: " + err.Message);
    }
}
I = I + 1
End While
dr.Close()
txtName.Text = " "
Catch ex As Exception
MsgBox(ex.ToString)
End Try
End Sub

Private Sub btnInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnInsert.Click
Try
Dim RowsInserted As Long
Dim cmd As ULCmd = Conn.CreateCommand()
cmd.CommandText = "INSERT INTO Names(name) VALUES (?)"
cmd.Parameters.Add("", txtName.Text)
RowsInserted = cmd.ExecuteNonQuery()
cmd.Dispose()
RefreshListBox()
Catch
MsgBox("Exception: " + Err.Description)
End Try
End Sub

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

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

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

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

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

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

- CultureInfo.CurrentUICulture
- CultureInfo.CurrentCulture
- EN

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

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.

**Namespace**

iAnywhere.Data.UltraLite
ULActiveSyncListener interface

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

**Visual Basic syntax**

```vbnet
Public Interface ULActiveSyncListener

**C# syntax**

```csharp
public interface ULActiveSyncListener
```

**Members**

All members of the ULActiveSyncListener interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveSyncInvoked</td>
<td>method Invoked when the MobiLink provider for ActiveSync calls the application to perform synchronization.</td>
</tr>
</tbody>
</table>

**ActiveSyncInvoked method**

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

**Visual Basic syntax**

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

**C# syntax**

```csharp
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 the ULDatabaseManager.SignalSyncIsComplete method to signal the MobiLink provider for ActiveSync.

See also

• “ULDatabaseManager.SignalSyncIsComplete method [UltraLite.NET]” on page 219

Example

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

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

    Private conn As ULConnection
    Public Sub New(ByVal args() As String)
        MyBase.New()
        ' This call is required by the Windows Form Designer.
        InitializeComponent()

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

        ' Create Connection
        ...
    End Sub

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

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

    Public Sub ActiveSyncAction( ByVal sender As Object, ByVal e As EventArgs _
    )
        ' Perform active sync.
        conn.Synchronize()
    End Sub
```
The following code is the C# language equivalent:

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

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

    public Form1()
    {
        // Required for Windows Form Designer support.
        // InitializeComponent();
        //
        // TODO: Add any constructor code after the
        // InitializeComponent call.
        //
        ULDatabaseManager.SetActiveSyncListener(
            "myCompany.myapp", this
        );
        // Create connection
        ...
    }

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

    protected override void OnClosing(
        System.ComponentModel.CancelEventArgs e )
    {
        ULDatabaseManager.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();
        ULDatabaseManager.SignalSyncIsComplete();
    }
}
```

**ULBulkCopy class**

Efficiently bulk loads an UltraLite table with data from another source.
**Visual Basic syntax**

Public NotInheritable Class ULBulkCopy Implements System.IDisposable

**C# syntax**

public sealed class ULBulkCopy : System.IDisposable

**Base classes**

- System.IDisposable

**Members**

All members of the ULBulkCopy class, including all inherited members.

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

**Remarks**

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

**ULBulkCopy constructor**

Initializes a ULBulkCopy object with the specified ULConnection object.
Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULBulkCopy(string) constructor</td>
<td>Initializes a ULBulkCopy object with the specified connection string.</td>
</tr>
<tr>
<td>ULBulkCopy(string, ULBulkCopyOptions) constructor</td>
<td>Initializes a ULBulkCopy object with the specified connection string, and copy options.</td>
</tr>
<tr>
<td>ULBulkCopy(ULConnection) constructor</td>
<td>Initializes a ULBulkCopy object with the specified ULConnection object.</td>
</tr>
<tr>
<td>ULBulkCopy(ULConnection, ULBulkCopyOptions, ULTransaction) constructor</td>
<td>Initializes a ULBulkCopy object with the specified ULConnection object, copy options, and ULTransaction object.</td>
</tr>
</tbody>
</table>

**ULBulkCopy(string) constructor**

Initializes a ULBulkCopy object with the specified connection string.

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

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

**Remarks**

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

This syntax opens a connection during a WriteToServer method call with the connectionString value. The connection is closed at the end of the WriteToServer call.

The connection string can be supplied using a ULConnectionParms object.

**See also**

- “ULConnection.ConnectionString property [UltraLite.NET]” on page 154
- “ULConnectionParms class [UltraLite.NET]” on page 163
- System.IDisposable
ULBulkCopy(string, ULBulkCopyOptions) constructor

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

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

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

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

**Remarks**

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

This syntax opens a connection during a WriteToServer method call with the connectionString value. The connection is closed at the end of the WriteToServer call.

**See also**

- “ULConnection.ConnectionString property [UltraLite.NET]” on page 154
- “ULBulkCopyOptions enumeration [UltraLite.NET]” on page 441

ULBulkCopy(ULConnection) constructor

Initializes a ULBulkCopy object with the specified ULConnection object.

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

- **connection**  The already open ULConnection object that is used to perform the bulk-copy operation. If the connection is not open, an exception is thrown during a WriteToServer method call.
Remarks

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

See also

- “ULConnection class [UltraLite.NET]” on page 118

ULBulkCopy(ULConnection, ULBulkCopyOptions, ULTransaction) constructor

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

Visual Basic syntax

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

C# syntax

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

Parameters

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

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

- **externalTransaction** An existing ULTransaction object under which the bulk copy occurs. If this value is not a null reference (Nothing in Visual Basic), then the bulk-copy operation is done within it. It is an error to specify both an external transaction and the ULBulkCopyOptions.UseInternalTransaction option.

Remarks

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

See also

- “ULConnection class [UltraLite.NET]” on page 118
- “ULTransaction class [UltraLite.NET]” on page 434
**Close method**

Closes the ULBulkCopy object.

**Visual Basic syntax**

```vbnet
Public Sub Close()
```

**C# syntax**

```csharp
public void Close()
```

**Dispose method**

Disposes of the ULBulkCopy object.

**Visual Basic syntax**

```vbnet
Public Sub Dispose()
```

**C# syntax**

```csharp
public void Dispose()
```

**WriteToServer method**

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

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WriteToServer(DataRow[])</code> method</td>
<td>Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName field of the ULBulkCopy object.</td>
</tr>
<tr>
<td><code>WriteToServer(DataTable)</code> method</td>
<td>Copies all rows in the supplied System.Data.DataTable to a destination table specified by the ULBulkCopy.DestinationTableName property.</td>
</tr>
<tr>
<td><code>WriteToServer(DataTable, DataRowState)</code> method</td>
<td>Copies all rows in the supplied System.Data.DataTable with the specified row state to a destination table specified by the ULBulkCopy.DestinationTableName property.</td>
</tr>
<tr>
<td><code>WriteToServer(IDataReader)</code> method</td>
<td>Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the ULBulkCopy.DestinationTableName property.</td>
</tr>
</tbody>
</table>
WriteToServer(DataRow[]) method
Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName field of the UBLBulkCopy object.

Visual Basic syntax
Public Sub WriteToServer(ByVal rows As DataRow())

C# syntax
public void WriteToServer(DataRow[] rows)

Parameters
● rows An array of System.Data.DataRow objects to be copied to the destination table.

See also
● “ULBulkCopy.DestinationTableName property [UltraLite.NET]” on page 55
● System.Data.DataRow

WriteToServer(DataTable) method
Copies all rows in the supplied System.Data.DataTable to a destination table specified by the UBLBulkCopy.DestinationTableName property.

Visual Basic syntax
Public Sub WriteToServer(ByVal table As DataTable)

C# syntax
public void WriteToServer(DataTable table)

Parameters
● table A System.Data.DataTable whose rows to be copied to the destination table.

See also
● “ULBulkCopy.DestinationTableName property [UltraLite.NET]” on page 55
● System.Data.DataTable

WriteToServer(DataTable, DataRowState) method
Copies all rows in the supplied System.Data.DataTable with the specified row state to a destination table specified by the UBLBulkCopy.DestinationTableName property.

Visual Basic syntax
Public Sub WriteToServer(
    ByVal table As DataTable,
    ByVal rowState As DataRowState
)
C# syntax

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

Parameters

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

Remarks

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

See also

- “ULBulkCopy.DestinationTableName property [UltraLite.NET]” on page 55
- System.Data.DataTable
- System.Data.DataRowState

**WriteToServer(IDataReader) method**

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

Visual Basic syntax

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

C# syntax

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

Parameters

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

See also

- “ULBulkCopy.DestinationTableName property [UltraLite.NET]” on page 55
- System.Data.IDataReader

**BatchSize property**

Gets or sets the number of rows in each batch.

Visual Basic syntax

```vbnet
Public Property BatchSize As Integer
```

C# syntax

```csharp
public int BatchSize {get;set;}
```
Remarks
At the end of each batch, the rows in the batch are sent to the server.

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

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

Setting it less than zero is an error.

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

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

Visual Basic syntax
Public Property BulkCopyTimeout As Integer

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

Remarks
The default value is 30 seconds.

Setting a value of zero indicates no limit, which 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 error is thrown.

Setting a value that is less than zero would throw an error.

ColumnMappings property
Returns a collection of ULBulkCopyColumnMapping items.

Visual Basic syntax
Public ReadOnly Property ColumnMappings As ULBulkCopyColumnMappingCollection

C# syntax
public ULBulkCopyColumnMappingCollection ColumnMappings {get;}

Remarks
Column mappings define the relationships between columns in the data source and columns in the destination.
By default, it is an empty collection.

The property cannot be modified while a WriteToServer method call is executing.

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

See also
- “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56

**DestinationTableName property**

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

**Visual Basic syntax**

```vbnet
Public Property DestinationTableName As String
```

**C# syntax**

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

**Remarks**

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

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

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

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

**NotifyAfter property**

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

**Visual Basic syntax**

```vbnet
Public Property NotifyAfter As Integer
```

**C# syntax**

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

**Remarks**

An integer representing the number of rows to be processed before generating a notification event, or zero is if the property has not been set.
Changes made to this property, while executing the WriteToServer method, do not take effect until after the next notification.

Setting this value to a value that is less than zero throws an error.

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

See also

- “ULBulkCopy.BulkCopyTimeout property [UltraLite.NET]” on page 54

**ULRowsCopied event**

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

**Visual Basic syntax**

```vbnet
Public Event ULRowsCopied As ULRowsCopiedEventHandler
```

**C# syntax**

```csharp
public event ULRowsCopiedEventHandler ULRowsCopied;
```

**Remarks**

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

See also

- “ULBulkCopy.NotifyAfter property [UltraLite.NET]” on page 55

**ULBulkCopyColumnMapping class**

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

**Visual Basic syntax**

```vbnet
Public NotInheritable Class ULBulkCopyColumnMapping
```

**C# syntax**

```csharp
public sealed class ULBulkCopyColumnMapping
```

**Members**

All members of the ULBulkCopyColumnMapping class, including all inherited members.
### ULBulkCopyColumnMapping class

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULBulkCopyColumnMapping constructor</td>
<td>Creates a new column mapping.</td>
</tr>
<tr>
<td>DestinationColumn property</td>
<td>Specifies the name of the column in the destination database table being mapped to.</td>
</tr>
<tr>
<td>DestinationOrdinal property</td>
<td>Specifies the ordinal value of the column in the destination database table being mapped to.</td>
</tr>
<tr>
<td>SourceColumn property</td>
<td>Specifies the name of the column being mapped in the data source.</td>
</tr>
<tr>
<td>SourceOrdinal property</td>
<td>Specifies the ordinal position of the source column within the data source.</td>
</tr>
</tbody>
</table>

**Remarks**

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

**See also**

- “ULBulkCopy class [UltraLite.NET]” on page 46

### ULBulkCopyColumnMapping constructor

Creates a new column mapping.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULBulkCopyColumnMapping() constructor</td>
<td>Creates a new column mapping.</td>
</tr>
<tr>
<td>ULBulkCopyColumnMapping(int, int) constructor</td>
<td>Creates a new column mapping, using column ordinals or names to refer to source and destination columns.</td>
</tr>
<tr>
<td>ULBulkCopyColumnMapping(int, string) constructor</td>
<td>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.</td>
</tr>
<tr>
<td>ULBulkCopyColumnMapping(string, int) constructor</td>
<td>Creates a new column mapping, using a column name to refer to the source column and a column ordinal to refer to the destination column.</td>
</tr>
<tr>
<td>ULBulkCopyColumnMapping(string, string) constructor</td>
<td>Creates a new column mapping, using column names to refer to source and destination columns.</td>
</tr>
</tbody>
</table>
ULBulkCopyColumnMapping() constructor

Creates a new column mapping.

Visual Basic syntax
Public Sub New()

C# syntax
public ULBulkCopyColumnMapping()

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

ULBulkCopyColumnMapping(int, int) constructor

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

Visual Basic syntax
Public Sub New(
    ByVal sourceColumnOrdinal As Integer,
    ByVal destinationColumnOrdinal As Integer
)

C# syntax
public ULBulkCopyColumnMapping(
    int sourceColumnOrdinal,
    int destinationColumnOrdinal
)

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

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

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

ULBulkCopyColumnMapping(int, string) constructor

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

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

C# syntax

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

Parameters

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

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

Remarks

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

**ULBulkCopyColumnMapping(string, int) constructor**

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

Visual Basic syntax

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

C# syntax

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

Parameters

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

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

Remarks

The ULBulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.
**ULBulkCopyColumnMapping(string, string) constructor**

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

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

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

**Remarks**

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

**DestinationColumn property**

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

**Visual Basic syntax**

```vbnet
Public Property DestinationColumn As String
```

**C# syntax**

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

**Remarks**

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

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

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

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

- “ULBulkCopyColumnMapping.DestinationOrdinal property [UltraLite.NET]” on page 61

**DestinationOrdinal property**

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

**Visual Basic syntax**

```vbnet
Public Property DestinationOrdinal As Integer
```

**C# syntax**

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

**Remarks**

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

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

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

See also

- “ULBulkCopyColumnMapping.DestinationColumn property [UltraLite.NET]” on page 60

**SourceColumn property**

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

**Visual Basic syntax**

```vbnet
Public Property SourceColumn As String
```

**C# syntax**

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

**Remarks**

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

The SourceColumn and SourceOrdinal properties are mutually exclusive. The most recently set value takes priority.
Setting the SourceColumn property causes the SourceOrdinal property to be set to -1. Setting the SourceOrdinal property causes the SourceColumn property to be set to a null reference (Nothing in Visual Basic).

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

See also

- “ULBulkCopyColumnMapping.SourceOrdinal property [UltraLite.NET]” on page 62

**SourceOrdinal property**

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

**Visual Basic syntax**

```vbnet
Public Property SourceOrdinal As Integer
```

**C# syntax**

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

**Remarks**

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

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

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

See also

- “ULBulkCopyColumnMapping.SourceColumn property [UltraLite.NET]” on page 61

**ULBulkCopyColumnMappingCollection class**

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

**Visual Basic syntax**

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

**C# syntax**

```csharp
public sealed class ULBulkCopyColumnMappingCollection : System.Collections.CollectionBase
```
## Base classes

- System.Collections.CollectionBase

## Members

All members of the ULBulkCopyColumnMappingCollection class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add method</td>
<td>Adds the specified ULBulkCopyColumnMapping object to the collection.</td>
</tr>
<tr>
<td>Clear method</td>
<td>Removes all objects from the System.Collections.CollectionBase instance.</td>
</tr>
<tr>
<td>Contains method</td>
<td>Returns whether the specified ULBulkCopyColumnMapping object exists in the collection.</td>
</tr>
<tr>
<td>CopyTo method</td>
<td>Copies the elements of the ULBulkCopyColumnMappingCollection object to an array of ULBulkCopyColumnMapping objects, starting at a particular index.</td>
</tr>
<tr>
<td>GetEnumerator method</td>
<td>Returns an enumerator that iterates through the System.Collections.CollectionBase instance.</td>
</tr>
<tr>
<td>IndexOf method</td>
<td>Returns the index of the specified ULBulkCopyColumnMapping object within the collection.</td>
</tr>
<tr>
<td>OnClear method</td>
<td>Performs additional custom processes when clearing the contents of the System.Collections.CollectionBase instance.</td>
</tr>
<tr>
<td>OnClearComplete method</td>
<td>Performs additional custom processes after clearing the contents of the System.Collections.CollectionBase instance.</td>
</tr>
<tr>
<td>OnInsert method</td>
<td>Performs additional custom processes before inserting a new element into the System.Collections.CollectionBase instance.</td>
</tr>
<tr>
<td>OnInsertComplete method</td>
<td>Performs additional custom processes after inserting a new element into the System.Collections.CollectionBase instance.</td>
</tr>
<tr>
<td>OnRemove method</td>
<td>Performs additional custom processes when removing an element from the System.Collections.CollectionBase instance.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>OnSet method</strong> (Inherited from System.Collections.CollectionBase)</td>
<td>Performs additional custom processes before setting a value in the System.Collections.CollectionBase instance.</td>
</tr>
<tr>
<td><strong>OnSetComplete method</strong> (Inherited from System.Collections.CollectionBase)</td>
<td>Performs additional custom processes after setting a value in the System.Collections.CollectionBase instance.</td>
</tr>
<tr>
<td><strong>OnValidate method</strong> (Inherited from System.Collections.CollectionBase)</td>
<td>Performs additional custom processes when validating a value.</td>
</tr>
<tr>
<td><strong>Remove method</strong></td>
<td>Removes the specified ULBulkCopyColumnMapping object from the ULBulkCopyColumnMappingCollection object.</td>
</tr>
<tr>
<td><strong>RemoveAt method</strong></td>
<td>Removes the mapping at the specified index from the collection.</td>
</tr>
<tr>
<td><strong>Capacity property</strong> (Inherited from System.Collections.CollectionBase)</td>
<td>Gets or sets the number of elements that the System.Collections.CollectionBase can contain.</td>
</tr>
<tr>
<td><strong>Count property</strong> (Inherited from System.Collections.CollectionBase)</td>
<td>Gets the number of elements contained in the System.Collections.CollectionBase instance.</td>
</tr>
<tr>
<td><strong>this property</strong></td>
<td>Gets the ULBulkCopyColumnMapping object at the specified index.</td>
</tr>
</tbody>
</table>

**Remarks**

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

**See also**

- “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56
Add method

Adds the specified ULBulkCopyColumnMapping object to the collection.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add(int, int) method</td>
<td>Creates a new ULBulkCopyColumnMapping object using ordinals to specify both source and destination columns, and adds the mapping to the collection.</td>
</tr>
<tr>
<td>Add(int, string) method</td>
<td>Creates a new ULBulkCopyColumnMapping object 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.</td>
</tr>
<tr>
<td>Add(string, int) method</td>
<td>Creates a new ULBulkCopyColumnMapping object using a column name to refer to the source column and a column ordinal to refer to the destination column, and adds the mapping to the collection.</td>
</tr>
<tr>
<td>Add(string, string) method</td>
<td>Creates a new ULBulkCopyColumnMapping object using column names to specify both source and destination columns, and adds the mapping to the collection.</td>
</tr>
<tr>
<td>Add(ULBulkCopyColumnMapping) method</td>
<td>Adds the specified ULBulkCopyColumnMapping object to the collection.</td>
</tr>
</tbody>
</table>

Add(int, int) method

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

Visual Basic syntax

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

C# syntax

```csharp
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.
The ordinal position of the destination column within the destination table. The first column in a table has ordinal position zero.

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

See also
● “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56

Add(int, string) method
Creates a new ULBulkCopyColumnMapping object using a column ordinal to refer to the source column and a column name to refer to the destination column, and adds mapping to the collection.

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

C# syntax
public ULBulkCopyColumnMapping Add(
    int sourceColumnOrdinal,
    string destinationColumn
)

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

● destinationColumn The name of the destination column within the destination table.

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

See also
● “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56

Add(string, int) method
Creates a new ULBulkCopyColumnMapping object using a column name to refer to the source column and a column ordinal to refer to the destination column, and adds the mapping to the collection.

Visual Basic syntax
Public Function Add(
    ByVal sourceColumn As String,

Add(string, string) method

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

Visual Basic syntax

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

C# syntax

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

Parameters

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

Remarks

The ULBulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.
See also
- “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56

Add(ULBulkCopyColumnMapping) method
Adds the specified ULBulkCopyColumnMapping object to the collection.

Visual Basic syntax
Public Function Add(
    ByVal bulkCopyColumnMapping As ULBulkCopyColumnMapping
) As ULBulkCopyColumnMapping

C# syntax
public ULBulkCopyColumnMapping Add(
    ULBulkCopyColumnMapping bulkCopyColumnMapping
)

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

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

See also
- “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56

Contains method
Returns whether the specified ULBulkCopyColumnMapping object exists in the collection.

Visual Basic syntax
Public Function Contains(
    ByVal value As ULBulkCopyColumnMapping
) As Boolean

C# syntax
public bool Contains(ULBulkCopyColumnMapping value)

Parameters
- value A valid ULBulkCopyColumnMapping object.

Returns
True if the specified mapping exists in the collection; otherwise, returns false.
See also

- “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56

**CopyTo method**

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

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

- **array** The one-dimensional ULBulkCopyColumnMapping array that is the destination of the elements copied from this ULBulkCopyColumnMappingCollection object. The array must have zero-based indexing.

- **index** The zero-based index in the array at which copying begins.

See also

- “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56

**IndexOf method**

Returns the index of the specified ULBulkCopyColumnMapping object within the collection.

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

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

**Returns**

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

Removes the specified ULBulkCopyColumnMapping object from the ULBulkCopyColumnMappingCollection object.

Visual Basic syntax

Public Sub Remove(ByVal value As ULBulkCopyColumnMapping)

C# syntax

public void Remove(ULBulkCopyColumnMapping value)

Parameters

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

See also

- “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56

RemoveAt method

Removes the mapping at the specified index from the collection.

Visual Basic syntax

Public Shadows Sub RemoveAt(ByVal index As Integer)

C# syntax

public new void RemoveAt(int index)

Parameters

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

this property

Gets the ULBulkCopyColumnMapping object at the specified index.

Visual Basic syntax

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

See also

- “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56
C# syntax

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

Parameters

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

Remarks

An ULBulkCopyColumnMapping object is returned.

See also

- “ULBulkCopyColumnMapping class [UltraLite.NET]” on page 56

**ULCommand class**

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

Visual Basic syntax

```vbnet
Public NotInheritable Class ULCommand
    Implements System.ICloneable
```

C# syntax

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

Base classes

- System.ICloneable

Members

All members of the ULCommand class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULCommand constructor</td>
<td>Initializes a ULCommand object.</td>
</tr>
<tr>
<td>BeginExecuteNonQuery method</td>
<td>Initiates the asynchronous execution of a SQL statement that is described by this ULCommand object.</td>
</tr>
<tr>
<td>BeginExecuteReader method</td>
<td>Initiates the asynchronous execution of a SQL statement that is described by this ULCommand object, and retrieves the result set.</td>
</tr>
<tr>
<td>Cancel method</td>
<td>This method is not supported in UltraLite.NET.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>CreateParameter method</td>
<td>Provides a ULParameter object for supplying parameters to ULCommand objects.</td>
</tr>
<tr>
<td>EndExecuteNonQuery method</td>
<td>Finishes asynchronous execution of a SQL statement.</td>
</tr>
<tr>
<td>EndExecuteReader method</td>
<td>Finishes asynchronous execution of a SQL statement, returning the requested ULDataReader.</td>
</tr>
<tr>
<td>ExecuteDbDataReader method</td>
<td>(Inherited from System.Data.Common.DbCommand) Executes the command text against the connection.</td>
</tr>
<tr>
<td>ExecuteNonQuery method</td>
<td>Executes a statement that does not return a result set, such as a SQL INSERT, DELETE, or UPDATE statement.</td>
</tr>
<tr>
<td>ExecuteReader method</td>
<td>Executes a SQL SELECT statement and returns the result set.</td>
</tr>
<tr>
<td>ExecuteResultSet method</td>
<td>UL Ext: Executes a SQL SELECT statement and returns the result set as a ULResultSet object.</td>
</tr>
<tr>
<td>ExecuteScalar method</td>
<td>Executes a SQL SELECT statement and returns a single value.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ExecuteTable method</td>
<td><strong>UL Ext:</strong> Retrieves a database table in a ULTable object for direct manipulation.</td>
</tr>
<tr>
<td>Prepare method</td>
<td>Pre-compiles and stores the SQL statement of this command.</td>
</tr>
<tr>
<td>CommandText property</td>
<td>Specifies the text of the SQL statement or the name of the table when the ULCommand.CommandType property is System.Data.CommandType/TableDirect.</td>
</tr>
<tr>
<td>CommandTimeout property</td>
<td>This feature is not supported by UltraLite.NET.</td>
</tr>
<tr>
<td>CommandType property</td>
<td>Specifies the type of command to be executed.</td>
</tr>
<tr>
<td>Connection property</td>
<td>The connection object on which to execute the ULCommand object.</td>
</tr>
<tr>
<td>DesignTimeVisible property</td>
<td>Indicates if the ULCommand object should be visible in a customized Windows Form Designer control.</td>
</tr>
<tr>
<td>IndexName property</td>
<td><strong>UL Ext:</strong> Specifies the name of the index to open (sort) the table with when the ULCommand.CommandType property is System.Data.CommandType/TableDirect.</td>
</tr>
<tr>
<td>Parameters property</td>
<td>Specifies the parameters for the current statement.</td>
</tr>
<tr>
<td>Plan property</td>
<td><strong>UL Ext:</strong> Returns the access plan UltraLite.NET uses to execute a query.</td>
</tr>
<tr>
<td>Transaction property</td>
<td>Specifies the ULTransaction object in which the ULCommand object executes.</td>
</tr>
<tr>
<td>UpdatedRowSource property</td>
<td>Specifies how command results are applied to the DataRow when used by the ULDataAdapterUpdate method.</td>
</tr>
</tbody>
</table>

**Remarks**

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

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

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

The ULCommand class features the following methods for executing commands on an UltraLite.NET database:
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULCommand.ExecuteNonQuery()</td>
<td>Executes a statement that does not return a result set, such as a SQL INSERT, DELETE, or UPDATE statement.</td>
</tr>
<tr>
<td>ULCommand.ExecuteReader()</td>
<td>Executes a SQL SELECT statement and returns the result set in a ULDataReader object. Use this method for creating read-only result sets.</td>
</tr>
<tr>
<td>ULCommand.ExecuteScalar()</td>
<td><strong>UL Ext:</strong> Executes a SQL SELECT statement and returns the result set in a ULResultSet object. Use this method for creating mutable result sets.</td>
</tr>
<tr>
<td>ULCommand.ExecuteTable()</td>
<td><strong>UL Ext:</strong> Retrieves a database table in a ULTable object for direct manipulation. The ULCommand.CommandText property is interpreted as the name of the table and the ULCommand.IndexName property can be used to specify a table sorting order. The ULCommand.CommandType property must be System.Data.CommandType.TableDirect.</td>
</tr>
</tbody>
</table>

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

See also

- “ULConnection.CreateCommand method [UltraLite.NET]” on page 132
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- “ULCommand.ExecuteNonQuery method [UltraLite.NET]” on page 91
- “ULCommand.ExecuteReader method [UltraLite.NET]” on page 92
- “ULCommand.ExecuteScalar method [UltraLite.NET]” on page 97
- “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
- “ULTable class [UltraLite.NET]” on page 401
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- “ULCommand.IndexName property [UltraLite.NET]” on page 104
- “ULResultSet class [UltraLite.NET]” on page 339
- “ULDataReader class [UltraLite.NET]” on page 228
- System.Data.IDbCommand
- System.IDisposable
- System.Data.CommandType
- System.ComponentModel.Component.Dispose

**ULCommand constructor**

Initializes a ULCommand object.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULCommand() constructor</td>
<td>Initializes a ULCommand object.</td>
</tr>
<tr>
<td>ULCommand(string) constructor</td>
<td>Initializes a ULCommand object with the specified command text.</td>
</tr>
<tr>
<td>ULCommand(string, ULConnection) constructor</td>
<td>Initializes a ULCommand object with the specified command text and connection.</td>
</tr>
<tr>
<td>ULCommand(string, ULConnection, ULTransaction) constructor</td>
<td>Initializes a ULCommand object with the specified command text, connection, and transaction.</td>
</tr>
</tbody>
</table>

**ULCommand() constructor**

Initializes a ULCommand object.

**Visual Basic syntax**

```
Public Sub New()
```
C# syntax

```csharp
public ULCommand()
```

Remarks

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

See also

- “ULConnection.CreateCommand method [UltraLite.NET]” on page 132
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.Connection property [UltraLite.NET]” on page 103
- “ULCommand.Transaction property [UltraLite.NET]” on page 106

**ULCommand(string) constructor**

Initializes a ULCommand object with the specified command text.

Visual Basic syntax

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

C# syntax

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

Parameters

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

Remarks

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

See also

- “ULConnection.CreateCommand method [UltraLite.NET]” on page 132
- “ULCommand.ULCommand constructor [UltraLite.NET]” on page 75
- “ULCommand.Connection property [UltraLite.NET]” on page 103
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- System.Data.CommandType

**ULCommand(string, ULConnection) constructor**

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

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

C# syntax

public ULCommand(string cmdText, ULConnection connection)

Parameters

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

- **connection**  The ULConnection object representing the current connection.

Remarks

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

See also

- “ULConnection class [UltraLite.NET]” on page 118
- “ULConnection.CreateCommand method [UltraLite.NET]” on page 132
- “ULCommand.ULCommand constructor [UltraLite.NET]” on page 75
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- System.Data.CommandType

**ULCommand(string, ULConnection, ULTransaction) constructor**

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

Visual Basic syntax

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

C# syntax

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

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

- **connection**  The ULConnection object representing the current connection.

- **transaction**  The ULTransaction object in which the ULCommand object executes.

See also

- “ULConnection.CreateCommand method [UltraLite.NET]” on page 132
- “ULCommand.ULCommand constructor [UltraLite.NET]” on page 75
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- “ULTransaction class [UltraLite.NET]” on page 434
- System.Data.CommandType

**BeginExecuteNonQuery method**

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

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeginExecuteNonQuery() method</td>
<td>Initiates the asynchronous execution of a SQL statement that is described by this ULCommand object.</td>
</tr>
<tr>
<td>BeginExecuteNonQuery(AsyncCallback, object) method</td>
<td>Initiates the asynchronous execution of a SQL statement that is described by this ULCommand object, given a callback procedure and state information.</td>
</tr>
</tbody>
</table>

**BeginExecuteNonQuery() method**

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

**Visual Basic syntax**

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

**C# syntax**

```csharp
public IAsyncResult BeginExecuteNonQuery()
```

**Returns**

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

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

See also

- “ULCommand.EndExecuteNonQuery method [UltraLite.NET]” on page 84
- System.IAsyncResult

**BeginExecuteNonQuery(AsyncCallback, object) method**

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

Visual Basic syntax

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

C# syntax

```csharp
public IAsyncResult BeginExecuteNonQuery(
    AsyncCallback callback,
    object stateObject
)
```

Parameters

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

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

Returns

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

Exceptions

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

See also

- “ULCommand.EndExecuteNonQuery method [UltraLite.NET]” on page 84
- System.IAsyncResult.AsyncState
- System.AsyncCallback
- System.IAsyncResult
**BeginExecuteReader method**

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

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeginExecuteReader() method</td>
<td>Initiates the asynchronous execution of a SQL statement that is described by this ULCommand object, and retrieves the result set.</td>
</tr>
<tr>
<td>BeginExecuteReader(AsyncCallback, object) method</td>
<td>Initiates the asynchronous execution of a SQL statement that is described by this ULCommand object, and retrieves the result set, given a callback procedure and state information.</td>
</tr>
<tr>
<td>BeginExecuteReader(AsyncCallback, object, CommandBehavior) method</td>
<td>Initiates the asynchronous execution of a SQL statement that is described by this ULCommand object, using one of the CommandBehavior values, and retrieves the result set, given a callback procedure and state information.</td>
</tr>
<tr>
<td>BeginExecuteReader(CommandBehavior) method</td>
<td>Initiates the asynchronous execution of a SQL statement that is described by this ULCommand object, using one of the CommandBehavior values, and retrieves the result set.</td>
</tr>
</tbody>
</table>

**BeginExecuteReader() method**

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

**Visual Basic syntax**

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

**C# syntax**

```csharp
public IAsyncResult BeginExecuteReader()
```

**Returns**

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

**Exceptions**

- **ULException class**  Any error that occurred while executing the command text.
BeginExecuteReader(AsyncCallback, object) method

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

Visual Basic syntax

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

C# syntax

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

Parameters

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

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

Returns

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

Exceptions

- **ULException class** Any error that occurred while executing the command text.
**BeginExecuteReader(AsyncCallback, object, CommandBehavior) method**

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

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

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

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


**Returns**

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

**Exceptions**

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

**See also**

- “ULCommand.EndExecuteReader method [UltraLite.NET]” on page 88
- “ULDataReader class [UltraLite.NET]” on page 228
- System.AsyncCallback
- System.IAsyncResult
- System.IAsyncResult.AsyncState
- System.Data.CommandBehavior
**BeginExecuteReader(CommandBehavior) method**
Initiates the asynchronous execution of a SQL statement that is described by this ULCommand object, using one of the CommandBehavior values, and retrieves the result set.

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

**C# syntax**
```
public IAsyncResult BeginExecuteReader(CommandBehavior cmdBehavior)
```

**Parameters**

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

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

**See also**
- “ULCommand.EndExecuteReader method [UltraLite.NET]” on page 88
- “ULDataReader class [UltraLite.NET]” on page 228
- System.Data.CommandBehavior
- System.IAsyncResult

**Cancel method**
This method is not supported in UltraLite.NET.

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

**C# syntax**
```
public override void Cancel()
```

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

Provides a ULParameter object for supplying parameters to ULCommand objects.

Visual Basic syntax

Public Shadows Function CreateParameter() As ULParameter

C# syntax

public new ULParameter CreateParameter()

Returns

A new parameter, as a ULParameter object.

Remarks

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


See also

● “ULParameter class [UltraLite.NET]” on page 307
● System.Data.IDbCommand.CreateParameter

EndExecuteNonQuery method

Finishes asynchronous execution of a SQL statement.

Visual Basic syntax

Public Function EndExecuteNonQuery( ByVal asyncResult As IAsyncResult ) As Integer

C# syntax

public int EndExecuteNonQuery(IAsyncResult asyncResult)

Parameters

● asyncResult The System.IAsyncResult returned by the call to the BeginExecuteNonQuery method.

Returns

The number of rows affected, which is the same behavior as the ExecuteNonQuery method.
Exceptions

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

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

Remarks

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

If an error occurs while executing the command, the exception is thrown when the EndExecuteNonQuery method 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
' Visual Basic
Dim cmd As ULCommand = New ULCommand(_
  "UPDATE Departments" _
  + " SET DepartmentName = 'Engineering'" _
  + " WHERE DepartmentID=100", _
  conn _
)
Dim res As IAsyncResult res = _
  cmd.BeginExecuteNonQuery()
'
' Perform other work.
' This blocks until the command completes.
Dim rowCount As Integer = _
  cmd.EndExecuteNonQuery( res )
```

The following code is the C# language equivalent:

```csharp
// C#
ULCommand cmd = new ULCommand(
  "UPDATE Departments"
  + " SET DepartmentName = 'Engineering'"
  + " WHERE DepartmentID=100",
  conn);
IAsyncResult res = cmd.BeginExecuteNonQuery();
// Perform other work.

// This blocks until the command completes.
int rowCount = cmd.EndExecuteNonQuery( res );
```
Poll the IsCompleted property of the IAsyncResult You can poll the IsCompleted property of the IAsyncResult. For example:

' Visual Basic
Dim cmd As ULCommand = new ULCommand(_
    "UPDATE Departments" _
    + " SET DepartmentName = 'Engineering'" _
    + " WHERE DepartmentID=100", _
    conn _
)
Dim res As IAsyncResult res = _
    cmd.BeginExecuteNonQuery()
While( !res.IsCompleted )
    ' Perform other work.
End While

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

The following code is the C# language equivalent:

// C#
ULCommand cmd = new ULCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'
    + " WHERE DepartmentID=100",
    conn);
IAsyncResult res = cmd.BeginExecuteNonQuery();
while( !res.IsCompleted ) {
    // Perform other work.
}

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

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

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

' Perform other work.

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

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

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

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

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

The following code is the C# language equivalent:

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

// Elsewhere in the code
private void DoStuff()
{
    ULCommand cmd = new ULCommand(
        "UPDATE Departments"
        + " SET DepartmentName = 'Engineering'"
        + " WHERE DepartmentID=100",
        conn
    );
    IAsyncResult res = cmd.BeginExecuteNonQuery(
        callbackFunction, cmd
    )
}
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.BeginExecuteNonQuery method [UltraLite.NET]” on page 78
- System.IAsyncResult

EndExecuteReader method

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

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

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

**Returns**

An ULDataReader object that can be used to retrieve the requested rows, which is the same behavior as the ExecuteReader method.

**Exceptions**

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

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

**Remarks**

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

If an error occurs while executing the command, the exception is thrown when the EndExecuteReader method is called.
There are four ways to wait for execution to complete:

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

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

The following code is the C# language equivalent:

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

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

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

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

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

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

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

// C#
ULCommand cmd = new ULCommand(
    "SELECT * FROM Departments", conn 
);
IAsyncResult res = cmd.BeginExecuteReader();
while( res.Status == AsyncStatus.Continuing ) {
    // Perform other work.
}
// This blocks until the command completes.
ULDataReader reader = cmd.EndExecuteReader( res );
```
"SELECT * FROM Departments", conn _
)
Dim res As IAsyncResult res = _
cmd.BeginExecuteReader()
' Perform other work.
Dim wh As WaitHandle = res.AsyncWaitHandle
wh.WaitOne()
' This does not block because the command is finished.
Dim reader As UDataReader = _
cmd.EndExecuteReader( res )

The following code is the C# language equivalent:

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

Specify a callback function when calling the BeginExecuteReader method You can specify a callback function when calling the BeginExecuteReader method. 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 UDataReader = cmd.EndExecuteReader()
  End Sub

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

The following code is the C# language equivalent:

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

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

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

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

See also
● “ULCommand.BeginExecuteReader method [UltraLite.NET]” on page 80
● “ULDataReader class [UltraLite.NET]” on page 228
● System.IAsyncResult

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

Visual Basic syntax
Public Overrides Function ExecuteNonQuery() As Integer

C# syntax
public override int ExecuteNonQuery() 

Returns
The number of rows affected.

Exceptions
● ULException class A SQL error occurred.

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

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

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

The ULCommand.CommandType property cannot be System.Data.CommandType.TableDirect.
See also

- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- “ULCommand.Connection property [UltraLite.NET]” on page 103
- “ULCommand.Parameters property [UltraLite.NET]” on page 105
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- System.Data.CommandType

**ExecuteReader method**

Executes a SQL SELECT statement and returns the result set.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteReader() method</td>
<td>Executes a SQL SELECT statement and returns the result set.</td>
</tr>
<tr>
<td>ExecuteReader(CommandBehavior) method</td>
<td>Executes a SQL SELECT statement with the specified command behavior and returns the result set.</td>
</tr>
</tbody>
</table>

**ExecuteReader() method**

Executes a SQL SELECT statement and returns the result set.

**Visual Basic syntax**

Public Shadows Function ExecuteReader() As ULDataReader

**C# syntax**

public new ULDataReader ExecuteReader()

**Returns**

The result set as a ULDataReader object.

**Exceptions**

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

**Remarks**

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

If the ULCommand.CommandType value is System.Data.CommandType.TableDirect, the ExecuteReader method performs a ULCommand.ExecuteTable call and returns a ULTable object downcast as a ULDataReader object.

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


**See also**
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- “ULCommand.Connection property [UltraLite.NET]” on page 103
- “ULCommand.Parameters property [UltraLite.NET]” on page 105
- “ULCommand.ExecuteResultSet method [UltraLite.NET]” on page 95
- “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- “ULDataAdapter class [UltraLite.NET]” on page 204
- “ULDataReader class [UltraLite.NET]” on page 228
- “ULTable class [UltraLite.NET]” on page 401
- System.Data.CommandType
- System.Data.IDbCommand.ExecuteReader

**ExecuteReader(CommandBehavior) method**

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

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

Returns

The result set as a ULDataReader object.

Exceptions

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

Remarks

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

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

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


See also

- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.Connection property [UltraLite.NET]” on page 103
- “ULCommand.ExecuteReader method [UltraLite.NET]” on page 92
- “ULCommand.ExecuteResultSet method [UltraLite.NET]” on page 95
- “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
- “ULCommand.Parameters property [UltraLite.NET]” on page 105
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- “ULDataAdapter class [UltraLite.NET]” on page 204
- “ULDataReader class [UltraLite.NET]” on page 228
- “ULTable class [UltraLite.NET]” on page 401
- System.Data.IDbCommand.ExecuteReader
- System.Data.CommandType
- System.Data.CommandBehavior
**ExecuteResultSet method**

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

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteResultSet() method</td>
<td><strong>UL Ext:</strong> Executes a SQL SELECT statement and returns the result set as a ULResultSet object.</td>
</tr>
<tr>
<td>ExecuteResultSet(CommandBehavior) method</td>
<td><strong>UL Ext:</strong> Executes a SQL SELECT statement with the specified command behavior and returns the result set as a ULResultSet object.</td>
</tr>
</tbody>
</table>

**ExecuteResultSet() method**

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

**Visual Basic syntax**

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

**C# syntax**

```csharp
public ULResultSet ExecuteResultSet()
```

**Returns**

The result set as a ULResultSet object.

**Exceptions**

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

**Remarks**

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

If the ULCommand.CommandType value is System.Data.CommandType.TableDirect, the ExecuteReader method performs a ULCommand.ExecuteTable call and returns a ULTTable object downcast as a ULResultSet object.

This method supports positioned updates and deletes with Dynamic SQL.
See also

- “ULCommand class [UltraLite.NET]” on page 71
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.Parameters property [UltraLite.NET]” on page 105
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- “ULResultSet class [UltraLite.NET]” on page 339
- “ULTable class [UltraLite.NET]” on page 401
- System.Data.CommandType

Example

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

ExecuteResultSet(CommandBehavior) method

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

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

- **cmdBehavior**  

**Returns**

The result set as a ULResultSet object.

**Exceptions**

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

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

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

This method supports positioned updates and deletes with Dynamic SQL.

See also
- “ULResultSet class [UltraLite.NET]” on page 339
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- “ULDataAdapter class [UltraLite.NET]” on page 204
- “ULTable class [UltraLite.NET]” on page 401
- System.Data.CommandType
- System.Data.CommandBehavior

ExecuteScalar method
Executes a SQL SELECT statement and returns a single value.

Visual Basic syntax
Public Overrides Function ExecuteScalar() As Object

C# syntax
public override object ExecuteScalar()

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

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

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

Remarks

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

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

If the ULCommand.CommandType value is System.Data.CommandType.TableDirect, the ExecuteScalar method performs a ULCommand.ExecuteTable call and returns the first column of the first row.

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

See also

- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- “ULCommand.Connection property [UltraLite.NET]” on page 103
- “ULCommand.IndexName property [UltraLite.NET]” on page 104
- “ULCommand.Parameters property [UltraLite.NET]” on page 105
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- System.Data.CommandType
- System.Data.CommandBehavior

**ExecuteTable method**

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

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteTable() method</td>
<td>UL Ext: Retrieves a database table in a ULTable object for direct manipulation.</td>
</tr>
<tr>
<td>ExecuteTable(CommandBehavior) method</td>
<td>UL Ext: Retrieves, with the specified command behavior, a database table for direct manipulation.</td>
</tr>
</tbody>
</table>
ExecuteTable() method

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

**Visual Basic syntax**

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

**C# syntax**

```csharp
public ULTable ExecuteTable()
```

**Returns**

The table as a ULTable object.

**Exceptions**

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

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

**Remarks**

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

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

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

**See also**

- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- “ULCommand.Connection property [UltraLite.NET]” on page 103
- “ULCommand.IndexName property [UltraLite.NET]” on page 104
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- “ULTable class [UltraLite.NET]” on page 401
- System.Data.CommandBehavior
- System.Data.CommandType

ExecuteTable(CommandBehavior) method

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

Public Function ExecuteTable (  
    ByVal cmdBehavior As CommandBehavior  
) As ULTable

C# syntax

public ULTable ExecuteTable(CommandBehavior cmdBehavior)

Parameters


Returns

The table as a ULTable object.

Exceptions

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

Remarks

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

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

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

See also

- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- “ULCommand.Connection property [UltraLite.NET]” on page 103
- “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
- “ULCommand.IndexName property [UltraLite.NET]” on page 104
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- System.Data.CommandBehavior
- System.Data.CommandType
Prepare method

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

Visual Basic syntax

Public Overrides Sub Prepare()

C# syntax

public override void Prepare()

Exceptions

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

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

Remarks

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

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

See also

- “ULCommand.Connection property [UltraLite.NET]” on page 103
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- “ULCommand.CommandText property [UltraLite.NET]” on page 101

CommandText property

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

Visual Basic syntax

Public Overrides Property CommandText As String

C# syntax

public override string CommandText {get;set;}

Remarks

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

A string specifying the text of the SQL statement or the name of the table. The default is an empty string (invalid command).
SELECT statements are marked as read-only by default for performance reasons. If the query is going to be used to make updates, the statement must end with "FOR UPDATE".

See also
- "ULCommand.ExecuteNonQuery method [UltraLite.NET]" on page 91
- "ULCommand.ExecuteReader method [UltraLite.NET]" on page 92
- "ULCommand.ExecuteResultSet method [UltraLite.NET]" on page 95
- "ULCommand.ExecuteScalar method [UltraLite.NET]" on page 97
- "ULCommand.ExecuteTable method [UltraLite.NET]" on page 98
- "ULCommand.CommandType property [UltraLite.NET]" on page 102
- System.Data.CommandType

Example

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

The following code is the C# language equivalent:

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

CommandTimeout property

This feature is not supported by UltraLite.NET.

Visual Basic syntax

    Public Overrides Property CommandTimeout As Integer

C# syntax

    public override int CommandTimeout {get;set;}

Exceptions

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

Remarks

The value is always zero.

CommandType property

Specifies the type of command to be executed.

Visual Basic syntax

    Public Overrides Property CommandType As CommandType

C# syntax

    public override CommandType CommandType {get;set;}
Exceptions

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

Remarks

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

Supported command types are as follows:

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

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

See also

- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.IndexName property [UltraLite.NET]” on page 104
- “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
- “ULCommand.ExecuteReader method [UltraLite.NET]” on page 92
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand.ExecuteNonQuery method [UltraLite.NET]” on page 91
- “ULCommand.ExecuteReader method [UltraLite.NET]” on page 92
- “ULCommand.ExecuteScalar method [UltraLite.NET]” on page 97
- System.Data.CommandType

**Connection property**

The connection object on which to execute the ULCommand object.

Visual Basic syntax

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

C# syntax

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

Remarks

The ULConnection object on which to execute the command.

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

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

See also
- “ULConnection class [UltraLite.NET]” on page 118
- System.Data.IDbCommand.Connection

**DesignTimeVisible property**
Indicates if the ULCommand object should be visible in a customized Windows Form Designer control.

**Visual Basic syntax**
```vbnet
Public Overrides Property DesignTimeVisible As Boolean
```

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

**Remarks**
True if this ULCommand object should be visible, false if this object should not be visible. The default is false.

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

**Visual Basic syntax**
```vbnet
Public Property IndexName As String
```

**C# syntax**
```csharp
public string IndexName {get;set;}
```

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

See also
- “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
- “ULCommand.ExecuteReader method [UltraLite.NET]” on page 92
- “ULCommand.CommandType property [UltraLite.NET]” on page 102
- System.Data.CommandType
**Parameters property**

Specifies the parameters for the current statement.

**Visual Basic syntax**

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

**C# syntax**

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

**Remarks**

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

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


**See also**

- “ULParameterCollection class [UltraLite.NET]” on page 321
- “ULParameter class [UltraLite.NET]” on page 307
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- System.Data.IDbCommand.Connection

**Plan property**

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

**Visual Basic syntax**

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

**C# syntax**

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

**Exceptions**

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

**Remarks**

This property is intended primarily for use during development.
A string containing the text-based description of the query execution plan.

**Transaction property**

Specifies the ULTransaction object in which the ULCommand object executes.

**Visual Basic syntax**

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

**C# syntax**

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

**Remarks**

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

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


**See also**

- “ULConnection.BeginTransaction method [UltraLite.NET]” on page 125
- “ULTransaction class [UltraLite.NET]” on page 434
- “ULCommand.Connection property [UltraLite.NET]” on page 103
- System.Data.IDbCommand.Transaction

**UpdatedRowSource property**

Specifies how command results are applied to the DataRow when used by the ULDataAdapterUpdate method.

**Visual Basic syntax**

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

**C# syntax**

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

**Remarks**

See also
  ● System.Data.UpdateRowSource

ULCommandBuilder class

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

Visual Basic syntax

```vbnet
Public Class ULCommandBuilder
```

C# syntax

```csharp
```

Base classes


Members

All members of the ULCommandBuilder class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULCommandBuilder constructor</td>
<td>Initializes a ULCommandBuilder object.</td>
</tr>
<tr>
<td>GetDeleteCommand method</td>
<td>Gets the automatically generated ULCommand object required to perform deletions on the database.</td>
</tr>
<tr>
<td>GetInsertCommand method</td>
<td>Gets the automatically generated ULCommand object required to perform insertions on the database.</td>
</tr>
<tr>
<td>GetParameterName method (Inherited from System.Data.Common.DbCommandBuilder)</td>
<td>Returns the name of the specified parameter in the format of @p#.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetUpdateCommand method</td>
<td>Gets the automatically generated ULCommand object required to perform updates on the database.</td>
</tr>
<tr>
<td>QuoteIdentifier method (Inherited from System.Data.Common.DbCommandBuilder)</td>
<td>Given an unquoted identifier in the correct catalog case, returns the correct quoted form of that identifier, including properly escaping any embedded quotes in the identifier.</td>
</tr>
<tr>
<td>UnquoteIdentifier method (Inherited from System.Data.Common.DbCommandBuilder)</td>
<td>Given a quoted identifier, returns the correct unquoted form of that identifier, including properly un-escaping any embedded quotes in the identifier.</td>
</tr>
</tbody>
</table>
Name | Description
--- | ---
**DataAdapter property** | Gets or sets a ULDataAdapter object for which SQL statements are automatically generated.

**QuotePrefix property** (Inherited from System.Data.Common.DbCommandBuilder) | Gets or sets the beginning character or characters to use when specifying database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.

**QuoteSuffix property** (Inherited from System.Data.Common.DbCommandBuilder) | Gets or sets the ending character or characters to use when specifying database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.

**SchemaSeparator property** (Inherited from System.Data.Common.DbCommandBuilder) | Gets or sets the character to be used for the separator between the schema identifier and any other identifiers.

**SetAllValues property** (Inherited from System.Data.Common.DbCommandBuilder) | Specifies whether all column values in an update statement are included or only changed ones.

**Remarks**

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

**See also**

- “ULCommand class [UltraLite.NET]” on page 71
- “ULConnection class [UltraLite.NET]” on page 118
- “ULDataAdapter class [UltraLite.NET]” on page 204
- System.Data.DataSet
- System.ComponentModel.Component
- System.IDisposable

**Example**

The following example uses the ULCommand object, along with the ULDataAdapter and ULConnection objects, 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 object.

```visualbasic
' 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 ULDaDataAdapter = New ULDaDataAdapter()
adapter.SelectCommand = New ULCmdCommand(queryString, connection)
Dim builder As ULCmdCommandBuilder = New ULCmdCommandBuilder(adapter)
connection.Open()
Dim dataSet As DataSet = New DataSet()
adapter.Fill(dataSet, tableName)
'Insert code to modify data in DataSet.
'Without the ULCmdCommandBuilder this line would fail
adapter.Update(dataSet, tableName)
Return dataSet
End Function

The following code is the C# language equivalent:

```c#
// C#
public static DataSet SelectULRows(string connectionString,
    string queryString, string tableName)
{
    using (ULConnection connection = new ULConnection(connectionString))
    {
        ULDaDataAdapter adapter = new ULDaDataAdapter();
        adapter.SelectCommand = new ULCmdCommand(queryString, connection);
        ULCmdCommandBuilder builder = new ULCmdCommandBuilder(adapter);
        connection.Open();
        DataSet dataSet = new DataSet();
        adapter.Fill(dataSet, tableName);
        // Insert code to modify data in DataSet.
        // Without the ULCmdCommandBuilder this line would fail
        adapter.Update(dataSet, tableName);
        return dataSet;
    }
}
```

### ULCmdCommandBuilder constructor

Initializes a ULCmdCommandBuilder object.

#### Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULCmdCommandBuilder() constructor</td>
<td>Initializes a ULCmdCommandBuilder object.</td>
</tr>
<tr>
<td>ULCmdCommandBuilder(ULDataAdap-ter) constructor</td>
<td>Initializes a ULCmdCommandBuilder object with the specified ULDaDataAdapter object.</td>
</tr>
</tbody>
</table>
**ULCommandBuilder() constructor**

Initializes a ULCommandBuilder object.

**Visual Basic syntax**

```vbnet
Public Sub New()
```

**C# syntax**

```csharp
public ULCommandBuilder()
```

**See also**

- “ULCommandBuilder.ULCommandBuilder constructor [UltraLite.NET]” on page 110

---

**ULCommandBuilder(ULDataAdapter) constructor**

Initializes a ULCommandBuilder object with the specified ULDataAdapter object.

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

- `adapter` A ULDataAdapter object.

**See also**

- “ULDataAdapter class [UltraLite.NET]” on page 204

---

**GetDeleteCommand method**

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

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDeleteCommand() method</td>
<td>Gets the automatically generated ULCommand object required to perform deletions on the database.</td>
</tr>
<tr>
<td>GetDeleteCommand(bool) method</td>
<td>Gets the automatically generated ULCommand object required to perform deletions on the database.</td>
</tr>
</tbody>
</table>
GetDeleteCommand() method

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

Visual Basic syntax

Public Shadows Function GetDeleteCommand() As ULCommand

C# syntax

public new ULCommand GetDeleteCommand()

Returns

The automatically generated ULCommand object required to perform deletions.

Exceptions

- InvalidOperationException  The DbCommandBuilder.DataAdapter 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 value that contains duplicate columns. Dynamic SQL generation for the DeleteCommand property is not supported against a SelectCommand value that does not return any key column information.

Remarks

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

See also

- “ULCommand class [UltraLite.NET]” on page 71
- “ULDataAdapter.SelectCommand property [UltraLite.NET]” on page 211

GetDeleteCommand(bool) method

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

Visual Basic syntax

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

C# syntax

public new ULCommand GetDeleteCommand(bool useColumnsForParameterNames)
Parameters

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

Returns

The automatically generated ULCommand object required to perform deletions.

Exceptions

- **InvalidOperationException**  
  The DbCommandBuilder.DataAdapter 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 value that contains duplicate columns. Dynamic SQL generation for the DeleteCommand property is not supported against a SelectCommand value that does not return any key column information.

Remarks

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

See also

- “ULCommand class [UltraLite.NET]” on page 71
- “ULDataAdapter.SelectCommand property [UltraLite.NET]” on page 211

### GetInsertCommand method

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

#### Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetInsertCommand() method</td>
<td>Gets the automatically generated ULCommand object required to perform insertions on the database.</td>
</tr>
<tr>
<td>GetInsertCommand(bool) method</td>
<td>Gets the automatically generated ULCommand object required to perform insertions on the database.</td>
</tr>
</tbody>
</table>
**GetInsertCommand() method**

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

**Visual Basic syntax**

```vbnet
Public Shadows Function GetInsertCommand() As ULCommand
```

**C# syntax**

```csharp
public new ULCommand GetInsertCommand()
```

**Returns**

The automatically generated ULCommand object required to perform insertions.

**Exceptions**

- `InvalidOperationException` The DbCommandBuilder.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 property is not supported against a SelectCommand value 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 value that contains duplicate columns.

**Remarks**

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

**See also**

- “ULCommand class [UltraLite.NET]” on page 71
- “ULDataAdapter.SelectCommand property [UltraLite.NET]” on page 211

---

**GetInsertCommand(bool) method**

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

**Visual Basic syntax**

```vbnet
Public Shadows Function GetInsertCommand(  
    ByVal useColumnsForParameterNames As Boolean  
) As ULCommand
```

**C# syntax**

```csharp
public new ULCommand GetInsertCommand(bool useColumnsForParameterNames)
```
Parameters

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

Returns

The automatically generated ULCommand object required to perform insertions.

Exceptions

- **InvalidOperationException**  The DbCommandBuilder.DataAdapter 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 property is not supported against a SelectCommand value 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 value that contains duplicate columns.

Remarks

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

See also

- “ULCommand class [UltraLite.NET]” on page 71
- “ULDataAdapter.SelectCommand property [UltraLite.NET]” on page 211

**GetUpdateCommand method**

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

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetUpdateCommand() method</td>
<td>Gets the automatically generated ULCommand object required to perform updates on the database.</td>
</tr>
<tr>
<td>GetUpdateCommand(bool) method</td>
<td>Gets the automatically generated ULCommand object required to perform updates on the database.</td>
</tr>
</tbody>
</table>
GetUpdateCommand() method

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

Visual Basic syntax

Public Shadows Function GetUpdateCommand() As ULCommand

C# syntax

public new ULCommand GetUpdateCommand()

Returns

The automatically generated ULCommand object required to perform updates.

Exceptions

- InvalidOperationException  
  The DbCommandBuilder.DataAdapter 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 property is not supported against a SelectCommand value 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 value that contains duplicate columns.
Dynamic SQL generation for the UpdateCommand property is not supported against a SelectCommand value that does not return any key column information.

Remarks

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

See also

- “ULCommand class [UltraLite.NET]” on page 71
- “ULDataAdapter.SelectCommand property [UltraLite.NET]” on page 211

GetUpdateCommand(bool) method

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

Visual Basic syntax

Public Shadows Function GetUpdateCommand(  
    ByVal useColumnsForParameterNames As Boolean  
) As ULCommand
public new ULCommand GetUpdateCommand(bool useColumnsForParameterNames)

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

Returns
The automatically generated ULCommand object required to perform updates.

Exceptions
● InvalidOperationException The DbCommandBuilder.DataAdapter 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 property is not supported against a SelectCommand value 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 value that contains duplicate columns. Dynamic SQL generation for the UpdateCommand property is not supported against a SelectCommand value that does not return any key column information.

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

See also
● “ULCommand class [UltraLite.NET]” on page 71
● “ULDataAdapter.SelectCommand property [UltraLite.NET]” on page 211

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

Visual Basic syntax
Public Shadows Property DataAdapter As ULD ataAdapter

C# syntax
public new ULD ataAdapter DataAdapter {get;set;}

Remarks
A ULD ataAdapter object.
See also

- “ULDataAdapter class [UltraLite.NET]” on page 204

## ULConnection class

Represents a connection to an UltraLite.NET database.

### Visual Basic syntax

```vbnet
Public NotInheritable Class ULConnection
    Inherits System.Data.Common.DbConnection
End Class
```

### C# syntax

```csharp
```

### Base classes


### Members

All members of the ULConnection class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULConnection constructor</td>
<td>Initializes a ULConnection object.</td>
</tr>
<tr>
<td><strong>BeginSynchronize</strong> method</td>
<td>UL Ext: Asynchronously launches a synchronization using the current the SyncParms object.</td>
</tr>
<tr>
<td><strong>BeginTransaction</strong> method</td>
<td>Returns a transaction object.</td>
</tr>
<tr>
<td><strong>CancelGetNotification</strong> method</td>
<td>UL Ext: Cancels any pending get-notification calls on all queues matching the given name.</td>
</tr>
<tr>
<td><strong>CancelSynchronize</strong> method</td>
<td>UL Ext: Causes a running synchronization to be canceled at the next opportunity.</td>
</tr>
<tr>
<td><strong>ChangeDatabase</strong> method</td>
<td>Changes the current database for an open connection.</td>
</tr>
<tr>
<td><strong>ChangeEncryptionKey</strong> method</td>
<td>UL Ext: Changes the database's encryption key to the specified new key.</td>
</tr>
<tr>
<td><strong>ChangePassword</strong> method</td>
<td>Changes the password for the user indicated in the connection string to the supplied new password.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Close method</td>
<td>Closes the database connection.</td>
</tr>
<tr>
<td>CountUploadRows method</td>
<td><strong>UL Ext:</strong> Returns the number of rows that need to be uploaded when the next synchronization takes place.</td>
</tr>
<tr>
<td>CreateCommand method</td>
<td>Creates and initializes a ULCommand object associated with this connection and its current transaction.</td>
</tr>
<tr>
<td>CreateNotificationQueue method</td>
<td><strong>UL Ext:</strong> Creates an event queue.</td>
</tr>
<tr>
<td>DeclareEvent method</td>
<td><strong>UL Ext:</strong> Declares a named event.</td>
</tr>
<tr>
<td>DestroyNotificationQueue method</td>
<td><strong>UL Ext:</strong> Destroys an event queue.</td>
</tr>
<tr>
<td>EndSynchronize method</td>
<td><strong>UL Ext:</strong> Blocks until an asynchronously launched synchronization terminates.</td>
</tr>
<tr>
<td>ExecuteTable method</td>
<td><strong>UL Ext:</strong> Retrieves a database table in a ULTable object for direct manipulation.</td>
</tr>
<tr>
<td>GetLastError method</td>
<td><strong>UL Ext:</strong> Returns the time of the most recent download of the specified publication.</td>
</tr>
<tr>
<td>GetNewUUID method</td>
<td><strong>UL Ext:</strong> Generates a new UUID (System.Guid).</td>
</tr>
<tr>
<td>GetNotification method</td>
<td><strong>UL Ext:</strong> Blocks for a notification or timeout.</td>
</tr>
<tr>
<td>GetNotificationParameter method</td>
<td><strong>UL Ext:</strong> Gets the value of a parameter for an event that was just read by the GetNotification method.</td>
</tr>
<tr>
<td>GetSchema method</td>
<td>Returns the list of supported schema collections.</td>
</tr>
<tr>
<td>GrantConnectTo method</td>
<td><strong>UL Ext:</strong> Grants access to an UltraLite database for a user ID with a specified password.</td>
</tr>
<tr>
<td>Open method</td>
<td>Opens a connection to a database using the previously-specified connection string.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>RegisterForEvent method</strong></td>
<td>UL Ext: Registers a queue to get events from an object.</td>
</tr>
<tr>
<td><strong>ResetLastDownloadTime method</strong></td>
<td>UL Ext: Resets the time of the most recent download.</td>
</tr>
<tr>
<td><strong>RevokeConnectFrom method</strong></td>
<td>UL Ext: Revokes access to an UltraLite database from the specified user ID.</td>
</tr>
<tr>
<td><strong>RollbackPartialDownload method</strong></td>
<td>UL Ext: Rolls back outstanding changes to the database from a partial download.</td>
</tr>
<tr>
<td><strong>SendNotification method</strong></td>
<td>UL Ext: Sends a notification to matching queues.</td>
</tr>
<tr>
<td><strong>SetSyncListener method</strong></td>
<td>Specifies the listener object used to process synchronization messages.</td>
</tr>
<tr>
<td><strong>StartSynchronizationDelete method</strong></td>
<td>UL Ext: Marks all subsequent deletes made by this connection for synchronization.</td>
</tr>
<tr>
<td><strong>StopSynchronizationDelete method</strong></td>
<td>UL Ext: Prevents delete operations from being synchronized.</td>
</tr>
<tr>
<td><strong>Synchronize method</strong></td>
<td>UL Ext: Synchronizes the database using the current ULConnection.SyncParms object.</td>
</tr>
<tr>
<td><strong>TriggerEvent method</strong></td>
<td>UL Ext: Triggers an event.</td>
</tr>
<tr>
<td><strong>ValidateDatabase method</strong></td>
<td>UL Ext: Performs validation on the current database.</td>
</tr>
<tr>
<td><strong>ConnectionString property</strong></td>
<td>Specifies the parameters to use for opening a connection to an UltraLite.NET database.</td>
</tr>
<tr>
<td><strong>ConnectionTimeout property</strong></td>
<td>This feature is not supported by UltraLite.NET.</td>
</tr>
<tr>
<td><strong>Database property</strong></td>
<td>Returns the name of the database to which the connection opens.</td>
</tr>
<tr>
<td><strong>DatabaseID property</strong></td>
<td>UL Ext: Specifies the Database ID value to be used for global autoincrement columns.</td>
</tr>
<tr>
<td><strong>DataSource property</strong></td>
<td>This feature is not supported by UltraLite.NET.</td>
</tr>
</tbody>
</table>
## ULConnection class

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalAutoIncrementUsage property</td>
<td><strong>UL Ext:</strong> Returns the percentage of available global auto-increment values that have been used.</td>
</tr>
<tr>
<td>LastIdentity property</td>
<td><strong>UL Ext:</strong> Returns the most recent identity value used.</td>
</tr>
<tr>
<td>Schema property</td>
<td><strong>UL Ext:</strong> Provides access to the schema of the current database associated with this connection.</td>
</tr>
<tr>
<td>ServerVersion property</td>
<td>This feature is not supported by UltraLite.NET.</td>
</tr>
<tr>
<td>State property</td>
<td>Returns the current state of the connection.</td>
</tr>
<tr>
<td>SyncParms property</td>
<td><strong>UL Ext:</strong> Specifies the synchronization settings for this connection.</td>
</tr>
<tr>
<td>SyncResult property</td>
<td><strong>UL Ext:</strong> Returns the results of the last synchronization for this connection.</td>
</tr>
<tr>
<td>InfoMessage event</td>
<td>Occurs when UltraLite.NET sends a warning or an informational message on this connection.</td>
</tr>
<tr>
<td>StateChange event</td>
<td>Occurs when this connection changes state.</td>
</tr>
<tr>
<td>INVALID_DATABASE_ID field</td>
<td><strong>UL Ext:</strong> A database ID constant indicating that the ULConnection.DatabaseID property has not been set.</td>
</tr>
<tr>
<td>SYNC_ALL_DB field</td>
<td>Empty publication list, corresponding to the entire database.</td>
</tr>
<tr>
<td>SYNC_ALL_PUBS field</td>
<td>Publication name &quot;*&quot;, corresponding to all publications.</td>
</tr>
</tbody>
</table>

### Remarks

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

A connection to an existing database is opened using the ULConnection.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 ULConnection.Schema value.
See also
- “ULConnection.Open method [UltraLite.NET]” on page 145
- “ULConnection.Schema property [UltraLite.NET]” on page 158
- System.Data.IDbConnection
- System.IDisposable

ULConnection constructor
Initializes a ULConnection object.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULConnection() constructor</td>
<td>Initializes a ULConnection object.</td>
</tr>
<tr>
<td>ULConnection(string) constructor</td>
<td>Initializes a ULConnection object with the specified connection string.</td>
</tr>
</tbody>
</table>

ULConnection() constructor
Initializes a ULConnection object.

Visual Basic syntax

Public Sub New()

C# syntax

public ULConnection();

Remarks
The connection must be opened before you can perform any operations against the database.

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

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

See also
- “ULConnection.Open method [UltraLite.NET]” on page 145
- “ULConnection.ConnectionString property [UltraLite.NET]” on page 154

ULConnection(string) constructor
Initializes a ULConnection object with the specified connection string.
**Visual Basic syntax**

    Public Sub New(ByVal connectionString As String)

**C# syntax**

    public ULConnection(string connectionString)

**Parameters**

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

**Exceptions**

- **ArgumentException** The supplied connection string is invalid.

**Remarks**

The connection must be opened before you can perform any operations against the database.

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

The connection string can be supplied using a ULConnectionParms object.

**See also**

- “ULConnection.Open method [UltraLite.NET]” on page 145
- “ULConnection.ULConnection constructor [UltraLite.NET]” on page 122
- “ULConnectionParms class [UltraLite.NET]” on page 163
- “ULConnection.ConnectionString property [UltraLite.NET]” on page 154

**Example**

The following code creates and opens a connection to the existing database \UltraLite\MyDatabase.udb on a Windows Mobile device.

    ' Visual Basic
    Dim openParms As ULConnectionParms = New ULConnectionParms
    openParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"
    Dim conn As ULConnection = _
        New ULConnection( openParms.ToString() )
    conn.Open()

The following code is the C# language equivalent:

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

**BeginSynchronize method**

UL Ext: Asynchronously launches a synchronization using the current the SyncParms object.
### Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeginSynchronize() method</td>
<td>UL Ext: Asynchronously launches a synchronization using the current the SyncParms object.</td>
</tr>
<tr>
<td>BeginSynchronize(Control, ULSyncProgressedDlg, object) method</td>
<td>UL Ext: Asynchronously launches a synchronization using the current SyncParms.</td>
</tr>
</tbody>
</table>

### BeginSynchronize() method

UL Ext: Asynchronously launches a synchronization using the current the SyncParms object.

**Visual Basic syntax**

```vbnet
Public Function BeginSynchronize() As IAsyncResult
```

**C# syntax**

```csharp
public IAsyncResult BeginSynchronize()
```

**Returns**

An IAsyncResult object that can be used to determine if the sync has completed or block until the sync has finished.

**Exceptions**

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

**Remarks**

This method will create a new thread to do the synchronization and then return immediately. Call the EndSynchronize method to block until the sync has completed.

**See also**

- “ULConnection.BeginSynchronize method [UltraLite.NET]” on page 123
- “ULConnection.EndSynchronize method [UltraLite.NET]” on page 134
- “ULConnection.CancelSynchronize method [UltraLite.NET]” on page 129
- “ULSyncProgressedDlg delegate [UltraLite.NET]” on page 439

### BeginSynchronize(Control, ULSyncProgressedDlg, object) method

UL Ext: Asynchronously launches a synchronization using the current SyncParms.

**Visual Basic syntax**

```vbnet
Public Function BeginSynchronize(
    ByVal control As Control,
    ByVal dlg As ULSyncProgressedDlg,
```
ByVal state As Object
) As IAsyncResult

C# syntax
public IAsyncResult BeginSynchronize(
    Control control,
    ULSyncProgressedDlg dlg,
    object state
)

Parameters
- **control**  A System.Windows.Forms.Control object the synchronization thread will use to invoke ULSyncProgressedDlg calls.
- **dlg**  A ULSyncProgressedDlg method that will be invoked regularly with synchronization progress updates.
- **state**  This user context can be accessed in the ULSyncProgressedDlg method using IAsyncResult.AsyncState.

Returns
An IAsyncResult object that can be used to determine if the sync has completed or block until the sync has finished.

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

Remarks
This method creates a new thread to do the synchronization and then return immediately, invoking the provided ULSyncProgressedDlg method regularly with synchronization progress updates. Call EndSynchronize to block until the sync has completed.

See also
- “ULConnection.BeginSynchronize method [UltraLite.NET]” on page 123
- “ULConnection.EndSynchronize method [UltraLite.NET]” on page 134
- “ULConnection.CancelSynchronize method [UltraLite.NET]” on page 129
- “ULSyncProgressedDlg delegate [UltraLite.NET]” on page 439

**BeginTransaction method**
Returns a transaction object.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeginTransaction() method</td>
<td>Returns a transaction object.</td>
</tr>
</tbody>
</table>
BeginTransaction(IsolationLevel) method

Returns a transaction object with the specified isolation level.

BeginTransaction() method

Returns a transaction object.

Visual Basic syntax

Public Shadows Function BeginTransaction() As ULTransaction

C# syntax

public new ULTransaction BeginTransaction()

Returns

A ULTransaction object representing the new transaction.

Exceptions

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

Remarks

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

The transaction is created with the IsolationLevel.ReadCommitted value.

To associate a command with a transaction object, use the ULCommand.Transaction property. The current transaction is automatically associated to commands created by the ULConnection.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 and the previous isolation level until the next call to BeginTransaction method.

UltraLite's definition of each isolation level is slightly different than ADO.NET's documentation of IsolationLevel.

See also

- “ULTransaction.Commit method [UltraLite.NET]” on page 435
- “ULTransaction.Rollback method [UltraLite.NET]” on page 436
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- “ULConnection.CreateCommand method [UltraLite.NET]” on page 132
- System.Data.IDbConnection.BeginTransaction
- “Isolation levels” [UltraLite - Database Management and Reference]

**BeginTransaction(IsolationLevel) method**

Returns a transaction object with the specified isolation level.

**Visual Basic syntax**

```vbnet
Public Shadows Function BeginTransaction(  
    ByVal isolationLevel As IsolationLevel  
) As ULTransaction
```

**C# syntax**

```csharp
public new ULTransaction BeginTransaction(IsolationLevel isolationLevel)
```

**Parameters**

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

**Returns**

A ULTransaction object representing the new transaction.

**Exceptions**

- **ULException class**  The connection is closed or an unsupported isolation level was specified.

- **InvalidOperationException**  The ULConnection class does not support parallel transactions.

**Remarks**

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

To associate a command with a transaction object, use the ULCommand.Transaction property. The current transaction is automatically associated to commands created by the ULConnection.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 and the previous isolation level until the next call to the BeginTransaction method.

UltraLite's definition of each isolation level is slightly different than ADO.NET's documentation of IsolationLevel.

See also
- “ULTransaction class [UltraLite.NET]” on page 434
- “ULConnection.BeginTransaction method [UltraLite.NET]” on page 125
- “ULTransaction.Commit method [UltraLite.NET]” on page 435
- “ULTransaction.Rollback method [UltraLite.NET]” on page 436
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- “ULConnection.CreateCommand method [UltraLite.NET]” on page 132
- System.Data.IDbConnection.BeginTransaction
- System.IsolationLevel
- “Isolation levels” [UltraLite - Database Management and Reference]

**CancelGetNotification method**

**UL Ext**: Cancels any pending get-notification calls on all queues matching the given name.

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

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

**Returns**

The number of affected queues (not the number of blocked reads necessarily).

**Exceptions**

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

**Remarks**

This method cancels any pending get-notification calls on all queues matching the given name.

**See also**

- “ULConnection.GetNotification method [UltraLite.NET]” on page 140
- “ULException class [UltraLite.NET]” on page 265
CancelSynchronize method

UL Ext: Causes a running synchronization to be canceled at the next opportunity.

Visual Basic syntax

Public Sub CancelSynchronize(ByVal asyncResult As IAsyncResult)

C# syntax

public void CancelSynchronize(IAsyncResult asyncResult)

Parameters

- asyncResult The IAsyncResult returned from the BeginSynchronize method.

Remarks

This method will inform the synchronization thread to terminate and returns immediately. Call the EndSynchronize method to block until the sync has successfully terminated.

See also

- “ULConnection.BeginSynchronize method [UltraLite.NET]” on page 123
- “ULConnection.EndSynchronize method [UltraLite.NET]” on page 134

ChangeDatabase method

Changes the current database for an open connection.

Visual Basic syntax

Public Overrides Sub ChangeDatabase(ByVal connectionString As String)

C# syntax

public override void ChangeDatabase(string connectionString)

Parameters

- connectionString A complete connection string to open the connection to a new database.

Remarks

The connection to the current database is closed even if there are parameter errors.

UL Ext: connectionString is a full connection string, not a dbn or dbf value.

See also

- “ULConnection.ConnectionString property [UltraLite.NET]” on page 154
ChangeEncryptionKey method

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

**Visual Basic syntax**

```vbnet
Public Sub ChangeEncryptionKey(ByVal newKey As String)
```

**C# syntax**

```csharp
public void ChangeEncryptionKey(string newKey)
```

**Parameters**

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

**Exceptions**

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

**Remarks**

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

**See also**

- “ULConnectionParms.EncryptionKey property [UltraLite.NET]” on page 170

ChangePassword method

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

**Visual Basic syntax**

```vbnet
Public Shared Sub ChangePassword(
    ByVal connectionString As String,
    ByVal newPassword As String
)
```

**C# syntax**

```csharp
public static void ChangePassword(
    string connectionString,
    string newPassword
)
```

**Parameters**

- **connectionString**  The connection string that contains enough information to connect to the database that you want. The connection string may contain the user ID and the current password.

- **newPassword**  The new password to set.

**Exceptions**

- **ArgumentNullException**  Either the connectionString or newPassword parameter is null.
**ArgumentException**  The connection string includes the option to use integrated security.

**ULException class**  A SQL error occurred while attempting to open the database.

## Close method
Closes the database connection.

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

**C# syntax**
```
public override void Close()
```

### Exceptions
- **ULException class**  A SQL error occurred.

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

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

**Visual Basic syntax**
```
Public Function CountUploadRows(
    ByVal pubs As String,
    ByVal threshold As Long
) As Long
```

**C# syntax**
```
public long CountUploadRows(string pubs, long threshold)
```

### Parameters
- **pubs**  A comma separated list of publications to check for rows.
- **threshold**  The maximum number of rows to count, limiting the amount of time taken by CountUploadRows. A value of 0 corresponds to the maximum limit. A value of 1 determines if any rows need to be synchronized.

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

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

### CreateCommand method

Creates and initializes a ULCommand object associated with this connection and its current transaction.

#### Visual Basic syntax

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

#### C# syntax

```
public new ULCommand CreateCommand()
```

#### Returns

A new ULCommand object.

#### Remarks

You can use the properties of the ULCommand object to control its behavior.

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


#### See also

- “ULCommand class [UltraLite.NET]” on page 71
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- System.Data.IDbConnection.CreateCommand

### CreateNotificationQueue method

**UL Ext:** Creates an event queue.

#### Visual Basic syntax

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

#### C# syntax

```
public void CreateNotificationQueue(string queueName, string parameters)
```

#### Parameters

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

**Exceptions**

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

**Remarks**

This method creates an event notification queue for this connection. Queue names are scoped per-connection, so different connections can create queues with the same name. When an event notification is sent, all queues in the database with a matching name receive (a separate instance of) the notification. Names are case insensitive. A default queue is created on demand for each connection when calling the RegisterForEvent event if no queue is specified.

**See also**

- “ULConnection.CreateNotificationQueue method [UltraLite.NET]” on page 132
- “ULException class [UltraLite.NET]” on page 265

---

**DeclareEvent method**

UL Ext: Declares a named event.

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

- **eventName**  
  The event name.

**Exceptions**

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

**Remarks**

Declare an event which can then be registered for and triggered. UltraLite predefines some system events triggered by operations on the database or the environment. The event name must be unique. Names are case insensitive. Throws error if name already used or invalid.

**See also**

- “ULConnection.CreateNotificationQueue method [UltraLite.NET]” on page 132
- “ULException class [UltraLite.NET]” on page 265

---

**DestroyNotificationQueue method**

UL Ext: Destroys an event queue.

---
Visual Basic syntax

Public Sub DestroyNotificationQueue(ByVal queueName As String)

C# syntax

public void DestroyNotificationQueue(string queueName)

Parameters

● queueName  The name of the queue.

Exceptions

● ULException class  A SQL error occurred.

Remarks

Destroy the given event notification queue. A warning is signaled if unread notifications remain in the queue. Unread notifications are discarded. A connection's default event queue, if created, is destroyed when the connection is closed.

See also

● “ULConnection.CreateNotificationQueue method [UltraLite.NET]” on page 132
● “ULException class [UltraLite.NET]” on page 265

EndSynchronize method

UL Ext: Blocks until an asynchronously launched synchronization terminates.

Visual Basic syntax

Public Sub EndSynchronize(ByVal asyncResult As IAsyncResult)

C# syntax

public void EndSynchronize(IAsyncResult asyncResult)

Parameters

● asyncResult  The IAsyncResult returned from the BeginSynchronize method.

Exceptions

● ULException class  A SQL error occurred.

Remarks

If an error occurred during the synchronization, a ULException exception is thrown.

See also

● “ULConnection.BeginSynchronize method [UltraLite.NET]” on page 123
● “ULConnection.CancelSynchronize method [UltraLite.NET]” on page 129
ExecuteTable method

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

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteTable(string) method</td>
<td>UL Ext: Retrieves a database table in a ULTable object for direct manipulation.</td>
</tr>
<tr>
<td>ExecuteTable(string, string) method</td>
<td>UL Ext: Retrieves a database table in a ULTable object for direct manipulation.</td>
</tr>
<tr>
<td>ExecuteTable(string, string, CommandBehavior) method</td>
<td>UL Ext: Retrieves, with the specified command behavior, a database table for direct manipulation.</td>
</tr>
</tbody>
</table>

ExecuteTable(string) method

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

Visual Basic syntax

Public Function ExecuteTable(ByVal tableName As String) As ULTable

C# syntax

public ULTable ExecuteTable(string tableName)

Parameters

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

Returns

The table as a ULTable object.

Exceptions

- `ULException class`  A SQL error occurred.

- `InvalidOperationException`  The `tableName` is invalid.

Remarks

The table is opened (sorted) using the table’s primary key.

This method is a shortcut for the ULCommand.ExecuteTable method that does not require a ULCommand object. It is provided to help users porting from earlier versions of UltraLite.NET (it replaces iAnywhere.UltraLite.Connection.GetTable and iAnywhere.UltraLite.Table.Open methods).
See also

- “ULConnection.ExecuteTable method [UltraLite.NET]” on page 135
- “ULTable class [UltraLite.NET]” on page 401
- “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
- “ULCommand class [UltraLite.NET]” on page 71

Example

The following code opens the table named MyTable using the table's primary key. It assumes an open ULConnection instance called conn.

' Visual Basic
Dim t As ULTable = conn.ExecuteTable("MyTable")

' The line above is equivalent to
' Dim cmd As ULCommand = conn.CreateCommand()
' cmd.CommandText = "MyTable"
' cmd.CommandType = CommandType.TableDirect
' Dim t As ULTable = cmd.ExecuteTable()
' cmd.Dispose()

The following code is the C# language equivalent:

// C#
ULTable t = conn.ExecuteTable("MyTable");

// The line above is equivalent to
// ULTable t;
// using(ULCommand cmd = conn.CreateCommand())
// {
//     cmd.CommandText = "MyTable";
//     cmd.CommandType = CommandType.TableDirect;
//     t = cmd.ExecuteTable();
// }

**ExecuteTable(string, string) method**

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

**Visual Basic syntax**

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

**C# syntax**

```
public ULTable ExecuteTable(string tableName, string indexName)
```

**Parameters**

- **tableName** The name of the table to open.
- **indexName** The name of the index with which to open (sort) the table.
Returns

The table as a ULTable object.

Exceptions

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

- **InvalidOperationException**
  The tableName is invalid.

Remarks

The table is opened (sorted) using the specified index.

This method is a shortcut for the ULCommand.ExecuteTable method that does not require a ULCommand object. It is provided to help users porting from earlier versions of UltraLite.NET (it replaces iAnywhere.UltraLite.Connection.GetTable and iAnywhere.UltraLite.Table.Open methods).

See also

- “ULConnection.ExecuteTable method [UltraLite.NET]” on page 135
- “ULTable class [UltraLite.NET]” on page 401
- “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
- “ULCommand class [UltraLite.NET]” on page 71

Example

The following code opens the table named MyTable using the index named MyIndex. It assumes an open ULConnection object called conn.

```visualbasic
' Visual Basic
Dim t As ULTable = conn.ExecuteTable("MyTable", "MyIndex")

' The line above is equivalent to
' Dim cmd As ULCommand = conn.CreateCommand()
' cmd.CommandText = "MyTable"
' cmd.IndexName = "MyIndex"
' cmd.CommandType = CommandType.TableDirect
' Dim t As ULTable = cmd.ExecuteTable()
' cmd.Dispose()
```

The following code is the C# language equivalent:

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

// The line above is equivalent to
// ULTable t;
// using(ULCommand cmd = conn.CreateCommand())
// {
//     cmd.CommandText = "MyTable";
//     cmd.IndexName = "MyIndex";
//     cmd.CommandType = CommandType.TableDirect;
//     t = cmd.ExecuteTable();
// }
```
ExecuteTable(string, string, CommandBehavior) method

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

Visual Basic syntax

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

C# syntax

public ULTable ExecuteTable(string tableName, string indexName, CommandBehavior cmdBehavior)

Parameters

● tableName The name of the table to open.
● indexName The name of the index with which to open (sort) the table.

Returns

The table as a ULTable object.

Exceptions

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

Remarks

The table is opened (sorted) using the specified index.

This method is a shortcut for the ULCommand.ExecuteTable(System.Data.CommandBehavior) method that does not require a ULCommand object. It is provided to help users porting from earlier versions of UltraLite.NET (it replaces iAnywhere.UltraLite.Connection.GetTable and iAnywhere.UltraLite.Table.Open methods).
See also

- “ULConnection.ExecuteTable method [UltraLite.NET]” on page 135
- “ULTable class [UltraLite.NET]” on page 401
- “ULCommand class [UltraLite.NET]” on page 71
- System.Data.CommandBehavior

Example

The following code opens the table named MyTable using the index named MyIndex. It assumes an open ULConnection object named conn.

```visualbasic
' Visual Basic
Dim t As ULTable = conn.ExecuteTable(  
    "MyTable", "MyIndex", CommandBehavior.Default  
)

' The line above is equivalent to the following code:
' Dim cmd As ULCommand = conn.CreateCommand()  
' cmd.CommandText = "MyTable"  
' cmd.IndexName = "MyIndex"  
' cmd.CommandType = CommandType.TableDirect  
' Dim t As ULTable = cmd.ExecuteTable(CommandBehavior.Default)  
' cmd.Dispose()
```

The following code is the C# language equivalent:

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

// The line above is equivalent to the following code:
// ULTable t;
// using(ULCommand cmd = conn.CreateCommand())
// {
//     cmd.CommandText = "MyTable";
//     cmd.IndexName = "MyIndex";
//     cmd.CommandType = CommandType.TableDirect;
//     t = cmd.ExecuteTable(CommandBehavior.Default);
// }
```

GetLastDownloadTime method

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

**Visual Basic syntax**

```vbnet
Public Function GetLastDownloadTime(ByVal publication As String) As Date
```

**C# syntax**

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

**Parameters**

- publication The publication to check.
Returns
The timestamp of the last download. If the SYNC_ALL_DB constant is used for publication, returns the
time of the last download of the entire database.

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

Remarks
The parameter *publication* is a publication name to check.

See also
- “ULConnection.ResetLastDownloadTime method [UltraLite.NET]” on page 147
- “ULConnection.SYNC_ALL_DB field [UltraLite.NET]” on page 163

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

Visual Basic syntax
```vbnet
Public Function GetNewUUID() As Guid
```

C# syntax
```csharp
public Guid GetNewUUID()
```

Returns
A new UUID as a System.Guid.

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

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

See also
- **System.Guid**

**GetNotification method**
UL Ext: Blocks for a notification or timeout.

Visual Basic syntax
```vbnet
Public Function GetNotification(
    ByVal queueName As String,
```

Without closing the code block.
ByVal wait_ms As Integer
    ) As String

C# syntax
    public string GetNotification(string queueName, int wait_ms)

Parameters
    ● queueName    The name of the queue to be waited upon.

    ● wait_ms       The time to wait, in milliseconds. Use System.Threading.Timeout.Infinite (-1) for an
                    indefinite wait.

Returns
    Null if wait period expired or was canceled; otherwise, returns the event name.

Exceptions
    ● ULException class    A SQL error occurred.

Remarks
    This method reads an event notification. This call blocks until a notification is received or until the given
    wait period expires. To wait indefinitely, pass System.Threading.Timeout.Infinite for the wait_ms
    parameter. To cancel a wait, send another notification to the given queue or use the CancelGetNotification
    method. After reading a notification, use the ReadNotificationParameter method to retrieve additional
    parameters.

See also
    ● “ULConnection.SendNotification method [UltraLite.NET]” on page 148
    ● “ULConnection.GetNotificationParameter method [UltraLite.NET]” on page 141
    ● “ULConnection.CancelGetNotification method [UltraLite.NET]” on page 128
    ● “ULException class [UltraLite.NET]” on page 265
    ● System.Threading.Timeout

GetNotificationParameter method

UL Ext: Gets the value of a parameter for an event that was just read by the GetNotification method.

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

C# syntax
    public string GetNotificationParameter(
        string queueName,
        string parameterName
    )
Parameters

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

Returns

The parameter value if the parameter was found; otherwise, returns null.

Exceptions

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

Remarks

This method get the value of a parameter for the event notification just read by the ULGetNotification method. Only the parameters from the most-recently read notification on the given queue are available.

See also

- “ULConnection.GetNotification method [UltraLite.NET]” on page 140
- “ULException class [UltraLite.NET]” on page 265

### GetSchema method

Returns the list of supported schema collections.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSchema() method</td>
<td>Returns the list of supported schema collections.</td>
</tr>
<tr>
<td>GetSchema(string) method</td>
<td>Returns information for the specified metadata collection for this ULConnection object.</td>
</tr>
<tr>
<td>GetSchema(string, string[]) method</td>
<td>Returns schema information for the data source of this ULConnection object and, if specified, uses the specified string for the schema name and the specified string array for the restriction values.</td>
</tr>
</tbody>
</table>

### GetSchema() method

Returns the list of supported schema collections.

**Visual Basic syntax**

```vbnet
Public Overrides Function GetSchema() As DataTable
```

**C# syntax**

```csharp
public override DataTable GetSchema()
```
GetSchema(string) method

Returns information for the specified metadata collection for this ULConnection object.

Visual Basic syntax

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

C# syntax

public override DataTable GetSchema(string collection)

Parameters

- **collection**  The name of the metadata collection. If no name is provided, the MetaDataCollections value is used.

See also

- “ULConnection.GetSchema method [UltraLite.NET]” on page 142
- “ULConnection class [UltraLite.NET]” on page 118

GetSchema(string, string[]) method

Returns schema information for the data source of this ULConnection object and, if specified, uses the specified string for the schema name and the specified string array for the restriction values.

Visual Basic syntax

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

C# syntax

public override DataTable GetSchema(
    string collection,
    string[] restrictions
)

Parameters

- **collection**  The name of the metadata collection. If no name is provided, the MetaDataCollections is used.

- **restrictions**  A set of restriction values for the requested schema.

Returns

A DataTable object that contains schema information.
Remarks

This method is used to query the database for various metadata. Each type of metadata is given a collection name, which must be passed to receive that data. The default collection name is MetaDataCollections.

You can query the .NET data provider to determine the list of supported schema collections by calling the GetSchema method with no arguments, or with the schema collection name MetaDataCollections. This returns a DataTable with a list of the supported schema collections (CollectionName), the number of restrictions that they each support (NumberOfRestrictions), and the number of identifier parts that they use (NumberOfIdentifierParts).

<table>
<thead>
<tr>
<th>Collection</th>
<th>Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>Returns information about all columns in the database.</td>
</tr>
<tr>
<td>DataSourceInformation</td>
<td>Returns information about the database provider.</td>
</tr>
<tr>
<td>DataTypes</td>
<td>Returns a list of supported data types.</td>
</tr>
<tr>
<td>ForeignKeys</td>
<td>Returns information about all foreign keys in the database.</td>
</tr>
<tr>
<td>IndexColumns</td>
<td>Returns information about all index columns in the database.</td>
</tr>
<tr>
<td>Indexes</td>
<td>Returns information about all indexes in the database.</td>
</tr>
<tr>
<td>MetaDataCollections</td>
<td>Returns a list of all collection names.</td>
</tr>
<tr>
<td>Publications</td>
<td>Returns information about all publications in the database.</td>
</tr>
<tr>
<td>ReservedWords</td>
<td>Returns a list of reserved words used by UltraLite.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>Returns information about restrictions used in GetSchema.</td>
</tr>
<tr>
<td>Tables</td>
<td>Returns information about all tables in the database.</td>
</tr>
</tbody>
</table>

These collection names are also available as read-only properties in the ULMetaDataCollectionNames class.

The results returned can be filtered by specifying an array of restrictions in the call to the GetSchema method.

The restrictions available with each collection can be queried by calling:

```csharp
GetSchema( "Restrictions" );
```

If the collection requires four restrictions, then the restrictions parameter must be either NULL, or a string with four values.

To filter on a particular restriction, place the string to filter by in its place in the array and leave any unused places NULL. For example, the Tables collection has three restrictions: Table, TableType, SyncType.
To filter the Table collection:

GetSchema( "Tables", new string[ ] { "my_table", NULL, NULL } ) Returns information about all tables named my_table.

GetSchema( "Tables", new string[ ] { NULL, "User", NULL } ) Returns information about all user tables.

See also
- “ULConnection class [UltraLite.NET]” on page 118
- “ULMetaDataCollectionNames class [UltraLite.NET]” on page 299

**GrantConnectTo method**

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

**Visual Basic syntax**

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

**C# syntax**

```
public void GrantConnectTo(string uid, string pwd)
```

**Parameters**

- **uid** The user ID to receive access to the database.
- **pwd** The password to be associated with the user ID.

**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**

- “ULConnectionParms.UserID property [UltraLite.NET]” on page 172
- “ULConnectionParms.Password property [UltraLite.NET]” on page 171
- “ULConnection.ConnectionString property [UltraLite.NET]” on page 154

**Open method**

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

**Visual Basic syntax**

```
Public Overrides Sub Open()
```
C# syntax

```csharp
public override void Open()
```

Exceptions

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

Remarks

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

See also

- “`ULConnection.ConnectionString` property [UltraLite.NET]” on page 154
- “`ULConnection.State` property [UltraLite.NET]” on page 159

RegisterForEvent method

**UL Ext:** Registers a queue to get events from an object.

Visual Basic syntax

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

C# syntax

```csharp
public void RegisterForEvent(
    string eventName,
    string objectName,
    string queueName,
    bool registerNotUnReg
)
```

Parameters

- `eventName` The event name.
- `objectName` The object name to which event applies. For example, a table name.
- `queueName` The event queue name to be used.
- `registerNotUnReg` True to register; false to unregister.

Exceptions

- `ULException class` A SQL error occurred.
Remarks
This method registers a queue to receive notifications of an event. The default connection queue is implied and created if a queue name is not supplied. Certain system events allow specification of an object name to which the event applies. For example, the TableModified event can specify the table name. Unlike the SendNotification method, only the specific queue registered receives notifications of the event; other queues with the same name on different connections do not (unless they are also explicit registered). This method throws an error if the queue or event does not exist.

See also
- “ULConnection.DeclareEvent method [UltraLite.NET]” on page 133
- “ULConnection.CreateNotificationQueue method [UltraLite.NET]” on page 132
- “ULException class [UltraLite.NET]” on page 265

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

Visual Basic syntax
Public Sub ResetLastDownloadTime(ByVal pubs As String)

C# syntax
public void ResetLastDownloadTime(string pubs)

Exceptions
- ULException class A SQL error occurred.

See also
- “ULConnection.GetLastDownloadTime method [UltraLite.NET]” on page 139

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

Visual Basic syntax
Public Sub RevokeConnectFrom(ByVal uid As String)

C# syntax
public void RevokeConnectFrom(string uid)

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

Exceptions
- ULException class A SQL error occurred.
See also
- “ULConnection.GrantConnectTo method [UltraLite.NET]” on page 145

RollbackPartialDownload method

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

**Visual Basic syntax**
```vbnet
Public Sub RollbackPartialDownload()
```

**C# syntax**
```csharp
public void RollbackPartialDownload()
```

**Exceptions**
- **ULEException class** A SQL error occurred.

See also
- “ULSyncParms.KeepPartialDownload property [UltraLite.NET]” on page 379
- “ULSyncParms.ResumePartialDownload property [UltraLite.NET]” on page 382

SendNotification method

**UL Ext:** Sends a notification to matching queues.

**Visual Basic syntax**
```vbnet
Public Function SendNotification(
    ByVal queueName As String,
    ByVal eventName As String,
    ByVal parameters As String
) As Integer
```

**C# syntax**
```csharp
public int SendNotification(
    string queueName,
    string eventName,
    string parameters
)
```

**Parameters**
- **queueName** The event queue name to be used.
- **eventName** The event name.
- **parameters** Parameters to pass.
Returns

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

Exceptions

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

Remarks

Returns the number of matching queues.

This method sends a notification to all queues matching the given name (including any such queue on the current connection). This call does not block. Use the special queue name "*" to send to all queues.

See also

- “ULConnection.DeclareEvent method [UltraLite.NET]” on page 133
- “ULConnection.RegisterForEvent method [UltraLite.NET]” on page 146
- “ULException class [UltraLite.NET]” on page 265

**SetSyncListener method**

Specifies the listener object used to process synchronization messages.

Visual Basic syntax

```vbnet
Public Sub SetSyncListener(ByVal listener As ULSyncProgressListener)
```

C# syntax

```csharp
public void SetSyncListener(ULSyncProgressListener listener)
```

Parameters

- **listener** The ULSyncProgressListener object that implements the SyncProgressed method, which is called for synchronization messages on this connection.

Exceptions

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

Remarks

When the SYNCHRONIZE profileName SQL statement is executed, its progress messages are routed to a syncListener object, if not null (Nothing in Visual Basic).

To remove the listener, pass a null reference in a call to the SetSyncListener method.

See also

- “ULSyncProgressListener interface [UltraLite.NET]” on page 396
StartSynchronizationDelete method

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

Visual Basic syntax
Public Sub StartSynchronizationDelete()

C# syntax
public void StartSynchronizationDelete()

Exceptions
- ULException class A SQL error occurred.

Remarks
When this method 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.StopSynchronizationDelete method [UltraLite.NET]” on page 150
- “ULTable.Truncate method [UltraLite.NET]” on page 420

StopSynchronizationDelete method

UL Ext: Prevents delete operations from being synchronized.

Visual Basic syntax
Public Sub StopSynchronizationDelete()

C# syntax
public void StopSynchronizationDelete()

Exceptions
- ULException class A SQL error occurred.

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

See also
- “ULConnection.StartSynchronizationDelete method [UltraLite.NET]” on page 150

Synchronize method

UL Ext: Synchronizes the database using the current ULConnection.SyncParms object.
Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Synchronize()</code> method</td>
<td><strong>UL Ext</strong>: Synchronizes the database using the current ULConnection.SyncParms object.</td>
</tr>
<tr>
<td><code>Synchronize(ULSyncProgressListener)</code></td>
<td><strong>UL Ext</strong>: Synchronizes the database using the current ULConnection.SyncParms object with progress events posted to the specified listener.</td>
</tr>
</tbody>
</table>

## `Synchronize()` method

**UL Ext**: Synchronizes the database using the current ULConnection.SyncParms object.

### Visual Basic syntax

```vbnet
Public Sub Synchronize()
```

### C# syntax

```csharp
public void Synchronize()
```

### Exceptions

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

### Remarks

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

### See also

- “ULConnection.Synchronize method [UltraLite.NET]” on page 150
- “ULConnection.SyncParms property [UltraLite.NET]” on page 160
- “ULConnection.SyncResult property [UltraLite.NET]” on page 160

## `Synchronize(ULSyncProgressListener)` method

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

### Visual Basic syntax

```vbnet
Public Sub Synchronize(ByVal listener As ULSyncProgressListener)
```

### C# syntax

```csharp
public void Synchronize(ULSyncProgressListener listener)
```

### Parameters

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

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

Remarks

Errors during synchronization are posted as a ULSyncProgressState.STATE_ERROR event, then thrown as a ULException.

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

See also

- “ULSyncProgressListener interface [UltraLite.NET]” on page 396
- “ULConnection.Synchronize method [UltraLite.NET]” on page 150
- “ULConnection.SyncParms property [UltraLite.NET]” on page 160
- “ULException class [UltraLite.NET]” on page 265
- “ULConnection.SyncResult property [UltraLite.NET]” on page 160

**TriggerEvent method**

**UL Ext**: Triggers an event.

**Visual Basic syntax**

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

**C# syntax**

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

**Parameters**

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

**Returns**

The number of event notifications sent.

**Exceptions**

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

**Remarks**

Returns the number of notifications sent.

This method triggers an event (and send notification to all registered queues).
See also

- “ULConnection.DeclareEvent method [UltraLite.NET]” on page 133
- “ULConnection.RegisterForEvent method [UltraLite.NET]” on page 146
- “ULException class [UltraLite.NET]” on page 265

**ValidateDatabase method**

UL Ext: Performs validation on the current database.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValidateDatabase(ULDBValid) method</td>
<td>UL Ext: Performs validation on the current database.</td>
</tr>
<tr>
<td>ValidateDatabase(ULDBValid, string) method</td>
<td>UL Ext: Performs validation on the current database.</td>
</tr>
</tbody>
</table>

**ValidateDatabase(ULDBValid) method**

UL Ext: Performs validation on the current database.

**Visual Basic syntax**

```vbnet
Public Sub ValidateDatabase(ByVal how As ULDBValid)
```

**C# syntax**

```csharp
public void ValidateDatabase(ULDBValid how)
```

**Parameters**

- **how** Describes how to validate the database.

**Exceptions**

- **ULException class** A SQLE_CORRUPT_ULTRALITE_INDEX or SQLE_CORRUPT_ULTRALITE_DATABASE error may occur if the database is corrupt.

**See also**

- “ULDatabaseManager.ValidateDatabase method [UltraLite.NET]” on page 220
- “ULDBValid enumeration [UltraLite.NET]” on page 441
- “Index validation failed for table %1, index %2 with code: %3” [Error Messages]
- “Database page validation failed with code: %1” [Error Messages]

**Example**

The following code validates the current database

```vbnet
' Visual Basic
conn.ValidateDatabase( iAnywhere.Data.UltraLite.ULVF_INDEX )
```
The following code is the C# language equivalent:

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

### ValidateDatabase(ULDBValid, string) method

**UL Ext:** Performs validation on the current database.

#### Visual Basic syntax

```vbnet
Public Sub ValidateDatabase(
    ByVal how As ULDBValid,
    ByVal tableName As String
)
```

#### C# syntax

```csharp
public void ValidateDatabase(ULDBValid how, string tableName)
```

#### Parameters

- **how** Describes how to validate the database.
- **tableName** If null (Nothing in Visual Basic), validate the entire database; otherwise, validate just the named table.

#### Exceptions

- **ULException class** A SQLE_CORRUPT_ULTRALITE_INDEX or SQLE_CORRUPT_ULTRALITE_DATABASE error may occur if the database is corrupt.

#### See also

- “ULDatabaseManager.ValidateDatabase method [UltraLite.NET]” on page 220
- “ULDBValid enumeration [UltraLite.NET]” on page 441
- “Index validation failed for table %1, index %2 with code: %3” [Error Messages]
- “Database page validation failed with code: %1” [Error Messages]

#### Example

The following code validates the current database

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

The following code is the C# language equivalent:

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

### ConnectionString property

Specifies the parameters to use for opening a connection to an UltraLite.NET database.
Visual Basic syntax

Public Overrides Property ConnectionString As String

C# syntax

public override string ConnectionString {get;set;}

Exceptions

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

Remarks

The connection string can be supplied using a ULConnectionParms object.

The parameters used to open this connection should be a semicolon-separated list of keyword-value pairs. The default is an empty string (an invalid connection string).

UL Ext: The parameters used by UltraLite.NET are specific to UltraLite databases and therefore the connection string is not compatible with SQL Anywhere connection strings.

Parameter values can be quoted with either single quote characters or double quote characters provided that the quoted contents do not contain quote characters of the same type. Values must be quoted if they contain semicolons, begin with a quote, or require leading or trailing whitespace.

If you are not quoting parameter values, make sure that they do not contain semicolons, and that they begin with either a single quote or a double quote character. Leading and trailing spaces in values are ignored.

By default, connections are opened with UID=DBA and PWD=sql. To make the database more secure, change the user DBA’s password or create new users (using the GrantConnectTo method) and remove the DBA user (using RevokeConnectFrom).

See also

- “ULConnection.Open method [UltraLite.NET]” on page 145
- “ULConnectionParms class [UltraLite.NET]” on page 163
- “ULConnection.GrantConnectTo method [UltraLite.NET]” on page 145
- “UltraLite connection parameters” [UltraLite - Database Management and Reference]

Example

The following code creates and opens a connection to the existing database \UltraLite\MyDatabase.udb on a Windows Mobile device.

' Visual Basic
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"
Dim conn As ULConnection = New ULConnection
conn.ConnectionString = openParms.ToString()
conn.Open()

The following code is the C# language equivalent:
// C#
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnDevice = @"\UltraLite\MyDatabase.udb";
ULConnection conn = new ULConnection();
conn.ConnectionString = openParms.ToString();
conn.Open();

**ConnectionTimeout property**
This feature is not supported by UltraLite.NET.

**Visual Basic syntax**
Public ReadOnly Overrides Property ConnectionTimeout As Integer

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

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

**Remarks**
The value is always zero.

**Database property**
Returns the name of the database to which the connection opens.

**Visual Basic syntax**
Public ReadOnly Overrides Property Database As String

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

**Remarks**
A string containing the name of the database.

On Windows Mobile devices, the ULConnection object looks in the connection string in the following order: dbn, ce_file.

On desktop machines, the ULConnection object looks in the connection string in the following order: dbn, nt_file.

**DatabaseID property**
UL Ext: Specifies the Database ID value to be used for global autoincrement columns.
Visual Basic syntax
    Public Property DatabaseID As Long

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

Exceptions
    ● ULException class  The specified new database ID is invalid.

Remarks
    The Database ID value of the current database.
    The database ID value must be in the range [0,System.UInt32.MaxValue]. A value of ULConnection.INVALID_DATABASE_ID is used to indicate that the database ID has not been set for the current database.

See also
    ● “ULDatabaseSchema.GetDatabaseProperty method [UltraLite.NET]” on page 222
    ● “ULDatabaseSchema.SetDatabaseOption method [UltraLite.NET]” on page 225
    ● “ULConnection.INVALID_DATABASE_ID field [UltraLite.NET]” on page 163
    ● System.UInt32.MaxValue

DataSource property
    This feature is not supported by UltraLite.NET.

Visual Basic syntax
    Public ReadOnly Overrides Property DataSource As String

C# syntax
    public override string DataSource {get;}

Remarks
    The value is always the empty string.

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

Visual Basic syntax
    Public ReadOnly Property GlobalAutoIncrementUsage As Short

C# syntax
    public short GlobalAutoIncrementUsage {get;}

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 157
Exceptions

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

Remarks

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

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

See also

- “ULDatabaseManager class [UltraLite.NET]” on page 214
- “ULConnection.DatabaseID property [UltraLite.NET]” on page 156

**LastIdentity property**

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

Visual Basic syntax

```
Public ReadOnly Property LastIdentity As ULong
```

C# syntax

```
public ulong LastIdentity {get;}
```

Exceptions

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

Remarks

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

The most recent identity value used. This property is equivalent to the SQL Anywhere statement:

```
SELECT @identity
```

The LastIdentity property 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, the LastIdentity property is one of the generated default values, but there is no reliable means to determine from which column the value is. For this reason, you should design your database and write your insert statements to avoid this situation.

**Schema property**

**UL Ext**: Provides access to the schema of the current database associated with this connection.
Visual Basic syntax
   Public Readonly Property Schema As ULDatabaseSchema

C# syntax
   public ULDatabaseSchema Schema {get;}

Remarks
   A reference to the ULDatabaseSchema object representing the schema of the database on which this connection opens.
   This property is only valid while its connection is open.

See also
   ● "ULDatabaseSchema class [UltraLite.NET]" on page 221

ServerVersion property
   This feature is not supported by UltraLite.NET.

Visual Basic syntax
   Public Readonly Overrides Property ServerVersion As String

C# syntax
   public override string ServerVersion {get;}

Remarks
   The value is always the empty string.

State property
   Returns the current state of the connection.

Visual Basic syntax
   Public Readonly Overrides Property State As ConnectionState

C# syntax
   public override ConnectionState State {get;}

Remarks
   Returns System.Data.ConnectionState.Open if the connection is open, or System.Data.ConnectionState.Closed if the connection is closed.
See also
- “ULConnection.StateChange event [UltraLite.NET]” on page 162
  - System.Data.ConnectionState

**SyncParms property**

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

**Visual Basic syntax**

```vbnet
Public ReadOnly Property SyncParms As ULSyncParms
```

**C# syntax**

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

**Remarks**

A reference to the ULSyncParms object representing the parameters used for synchronization by this connection. Modifications to the parameters affect the next synchronization made over this connection.

See also
- “ULConnection.Synchronize method [UltraLite.NET]” on page 150
- “ULConnection.SyncResult property [UltraLite.NET]” on page 160
- “ULSyncParms class [UltraLite.NET]” on page 375

**SyncResult property**

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

**Visual Basic syntax**

```vbnet
Public ReadOnly Property SyncResult As ULSyncResult
```

**C# syntax**

```csharp
public ULSyncResult SyncResult {get;}
```

**Remarks**

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

See also
- “ULConnection.Synchronize method [UltraLite.NET]” on page 150
- “ULConnection.SyncParms property [UltraLite.NET]” on page 160
- “ULConnection.SyncResult property [UltraLite.NET]” on page 160
InfoMessage event

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

Visual Basic syntax

Public Event InfoMessage As ULInfoMessageEventHandler

C# syntax

public event ULInfoMessageEventHandler InfoMessage;

Remarks

To process UltraLite.NET warnings or informational messages, you must create a ULInfoMessageEventHandler delegate and attach it to this event.

See also

● “ULInfoMessageEventHandler delegate [UltraLite.NET]” on page 437

Example

The following code defines an informational message event handler:

' Visual Basic
Private Sub MyInfoMessageHandler( _
    obj As Object, args As ULInfoMessageEventArgs _
)
    System.Console.WriteLine( _
        "InfoMessageHandler: " + args.NativeError + ", " _
        + args.Message _
    )
End Sub

The following code is the C# language equivalent:

// C#
private void MyInfoMessageHandler(
    object obj, ULInfoMessageEventArgs args
)
{
    System.Console.WriteLine(
        "InfoMessageHandler: " + args.NativeError + ", " _
        + args.Message
    );
}

The following code adds the MyInfoMessageHandler method to the connection named conn.

' Visual Basic
AddHandler conn.InfoMessage, AddressOf MyInfoMessageHandler

The following code is the C# language equivalent:

// C#
conn.InfoMessage +=
    new ULInfoMessageEventHandler(MyInfoMessageHandler);
StateChange event

Occurs when this connection changes state.

Visual Basic syntax

Public Event StateChange As StateChangeEventHandler

C# syntax

public event override StateChangeEventHandler StateChange;

Remarks

To process state change messages, you must create a System.Data.StateChangeEventHandler delegate and attach it to this event.

See also

● System.Data.StateChangeEventHandler

Example

The following code defines a state change event handler.

' Visual Basic
Private Sub MyStateHandler( _
    obj As Object, args As StateChangeEventArgs _
)________________________
    System.Console.WriteLine( _
        "StateHandler: " + args.OriginalState + " to " + args.CurrentState _
    )________________________
End Sub

The following code is the C# language equivalent:

// C#
private void MyStateHandler( _
    object obj, StateChangeEventArgs args _
)________________________
    System.Console.WriteLine( _
        "StateHandler: " + args.OriginalState + " to " + args.CurrentState _
    );________________________

The following code adds the MyStateHandler to the connection named conn.

' Visual Basic
AddHandler conn.StateChange, AddressOf MyStateHandler

The following code is the C# language equivalent:

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

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

**Visual Basic syntax**

```
Public Const INVALID_DATABASE_ID As Long
```

**C# syntax**

```
public const long INVALID_DATABASE_ID;
```

**See also**

- “ULConnection.DatabaseID property [UltraLite.NET]” on page 156

SYNC_ALL_DB field

Empty publication list, corresponding to the entire database.

**Visual Basic syntax**

```
Public Const SYNC_ALL_DB As String
```

**C# syntax**

```
public const String SYNC_ALL_DB;
```

SYNC_ALL_PUBS field

Publication name "*", corresponding to all publications.

**Visual Basic syntax**

```
Public Const SYNC_ALL_PUBS As String
```

**C# syntax**

```
public const String SYNC_ALL_PUBS;
```

ULConnectionParms class

**UL Ext**: Builds a connection string for opening a connection to an UltraLite database.

**Visual Basic syntax**

```
Public Class ULConnectionParms Inherits System.ComponentModel.Component
```

**C# syntax**

```
public class ULConnectionParms : System.ComponentModel.Component
```
# Base classes
- System.ComponentModel.Component

# Members
All members of the ULConnectionParms class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULConnectionParms constructor</td>
<td>Initializes a ULConnectionParms instance with its default values.</td>
</tr>
<tr>
<td>Dispose method (Inherited from System.ComponentModel.Component)</td>
<td>Releases all resources used by the System.ComponentModel.Component.</td>
</tr>
<tr>
<td>Finalize method (Inherited from System.ComponentModel.Component)</td>
<td>Releases unmanaged resources and performs other cleanup operations before the System.ComponentModel.Component is reclaimed by garbage collection.</td>
</tr>
<tr>
<td>GetService method (Inherited from System.ComponentModel.Component)</td>
<td>Returns an object that represents a service provided by the System.ComponentModel.Component or by its System.ComponentModel.IContainer.</td>
</tr>
<tr>
<td>ToString method</td>
<td>Returns the string representation of this instance.</td>
</tr>
<tr>
<td>AdditionalParms property</td>
<td>Specifies additional parameters as a semicolon-separated list of name=value pairs.</td>
</tr>
<tr>
<td>CacheSize property</td>
<td>Specifies the size of the cache.</td>
</tr>
<tr>
<td>CanRaiseEvents property (Inherited from System.ComponentModel.Component)</td>
<td>Gets a value indicating whether the component can raise an event.</td>
</tr>
<tr>
<td>ConnectionName property</td>
<td>Specifies a name for the connection.</td>
</tr>
<tr>
<td>DatabaseOnDesktop property</td>
<td>Specifies the path and file name of the UltraLite database on Windows desktop platforms.</td>
</tr>
<tr>
<td>DatabaseOnDevice property</td>
<td>Specifies the path and file name of the UltraLite database on Windows Mobile.</td>
</tr>
<tr>
<td>DesignMode property (Inherited from System.ComponentModel.Component)</td>
<td>Gets a value that indicates whether the System.ComponentModel.Component is currently in design mode.</td>
</tr>
<tr>
<td>EncryptionKey property</td>
<td>Specifies a key for encrypting the database.</td>
</tr>
</tbody>
</table>
**Name** | **Description**
--- | ---
Disposed (Inherited from System.ComponentModel.Component) | Gets the list of event handlers that are attached to this System.ComponentModel.Component. Occurs when the component is disposed by a call to the System.ComponentModel.Component.Dispose method.

Events property (Inherited from System.ComponentModel.Component) | 

Password property | Specifies the password for the authenticated user.


UserID property | Specifies an authenticated user for the database.

**Remarks**

The frequently-used connection parameters are individual properties on the ULConnectionParms object.

A ULConnectionParms object is used to specify the parameters for opening a connection (with the ULConnection.Open method) or dropping a database (with the ULDatabaseManager.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 ULConnectionParms.ToString method. The resulting string is used to create a new ULConnection object with the ULConnection(String) constructor or set the ULConnection.ConnectionString property of an existing ULConnection object.

**Identifying the database**

Each instance contains platform-specific paths to the database. Only the value corresponding to the executing platform is used. For example, in the code below the path \UltraLite\mydb1.udb would be used on Windows Mobile, while mydb2.db would be used on other platforms.

' Visual Basic
Dim dbName As ULConnectionParms = new ULConnectionParms
dbName.DatabaseOnDevice = "\UltraLite\mydb1.udb"
dbName.DatabaseOnDesktop = "somedir\mydb2.udb"

The following code is the C# language equivalent:

// C#
ULConnectionParms db_name = new ULConnectionParms();
dbName.DatabaseOnDevice = "\UltraLite\mydb1.udb";
dbName.DatabaseOnDesktop = "somedir\mydb2.udb";
The recommended extension for UltraLite database files is .udb. On Windows Mobile devices, the default database is \UltraLiteDB\ulstore.udb. On other Windows platforms, the default database is ulstore.udb. In C#, you must escape any backslash characters in paths or use @-quoted string literals.

If you are using multiple databases, you must specify a database name for each database.

Optional connection settings

Depending on your application's needs and how the database was created, you might need to supply a non-default ULConnectionParms.UserID value, a non-default ULConnectionParms.Password value, a database ULConnectionParms.EncryptionKey value, and the ULConnectionParms.CacheSize value. If your application is using multiple connections, you should provide a unique ULConnectionParms.ConnectionName value for each connection.

Databases are created with a single authenticated user, DBA, whose initial password is sql. By default, connections are opened using the user ID DBA and password sql. To disable the default user, use the ULConnection.RevokeConnectFrom method. To add a user or change a user's password, use the ULConnection.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 ULConnection.ChangeEncryptionKey method.

See also

- “ULConnection.Open method [UltraLite.NET]” on page 145
- “ULDatabaseManager.DropDatabase method [UltraLite.NET]” on page 216
- “ULConnectionParms.ToString method [UltraLite.NET]” on page 167
- “ULConnection class [UltraLite.NET]” on page 118
- “ULConnection.ConnectionString property [UltraLite.NET]” on page 154
- “ULConnectionParms.UserID property [UltraLite.NET]” on page 172
- “ULConnectionParms.Password property [UltraLite.NET]” on page 171
- “ULConnectionParms.EncryptionKey property [UltraLite.NET]” on page 170
- “ULConnectionParms.CacheSize property [UltraLite.NET]” on page 168
- “ULConnectionParms.ConnectionName property [UltraLite.NET]” on page 169
- “ULConnectionParms.AdditionalParms property [UltraLite.NET]” on page 167
- “ULConnection.RevokeConnectFrom method [UltraLite.NET]” on page 147
- “ULConnection.GrantConnectTo method [UltraLite.NET]” on page 145
- “ULConnection.ChangeEncryptionKey method [UltraLite.NET]” on page 130
- “UltraLite connection parameters” [UltraLite - Database Management and Reference]

ULConnectionParms constructor

Initializes a ULConnectionParms instance with its default values.

Visual Basic syntax

Public Sub New()
C# syntax
    public ULConnectionParms()

**ToString method**

Returns the string representation of this instance.

Visual Basic syntax
    Public Overrides Function ToString() As String

C# syntax
    public override string ToString()

**Returns**

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

**AdditionalParms property**

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

Visual Basic syntax
    Public Property AdditionalParms As String

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

**Exceptions**

- ArgumentException  The value contained an invalid connection string.

**Remarks**

These parameters are used less frequently.

A semicolon-separated list of keyword=value additional parameters. Values of the keyword=value list must conform to the rules for ULConnection.ConnectionString. The default is a null reference (Nothing in Visual Basic).

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

Additional parameters are:
<table>
<thead>
<tr>
<th><strong>Keyword</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>dbn</td>
<td>Identifies a loaded database to which a connection needs to be made. When a database is started, it is assigned a database name, either explicitly with the dbn parameter, or by UltraLite using the base of the file name with the extension and path removed. When opening connections, UltraLite first searches for a running database with a matching dbn value. If one is not found, UltraLite starts a new database using the appropriate database file name parameter (with the DatabaseOnDevice or DatabaseOnDesktop properties). This parameter is required if the application (or UltraLite engine) needs to access two different databases that have the same base file name. This parameter is only used when opening a connection with the ULConnection.Open method.</td>
</tr>
<tr>
<td>reserve_size</td>
<td>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. The reserve_size parameter reserves file system space, which includes the metadata in the persistent store file, and not just the raw data. The metadata overhead and data compression must be considered when deriving the required file system space from the amount of database data. The reserve_size parameter reserves space by growing the persistent store file to the given reserve size on startup, regardless of whether the file previously existed. The file is never truncated. The following parameter string ensures that the persistent store file is at least 2 MB upon startup: createParms.AdditionalParms = &quot;reserve_size=2m&quot; This parameter is only used when opening a connection with the ULConnection.Open method.</td>
</tr>
<tr>
<td>start</td>
<td>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.</td>
</tr>
</tbody>
</table>

See also
- “ULDatabaseManager.RuntimeType property [UltraLite.NET]” on page 220
- “ULConnection.ConnectionString property [UltraLite.NET]” on page 154
- “ULConnectionParms.DatabaseOnDevice property [UltraLite.NET]” on page 170
- “ULConnectionParms.DatabaseOnDesktop property [UltraLite.NET]” on page 170
- “ULConnection.Open method [UltraLite.NET]” on page 145
- “UltraLite connection parameters” [UltraLite - Database Management and Reference]

**CacheSize property**

Specifies the size of the cache.

**Visual Basic syntax**

```
Public Property CacheSize As String
```
C# syntax

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

Exceptions

- ArgumentException The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

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

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

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

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

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

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

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

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

---

**ConnectionName property**

Specifies a name for the connection.

Visual Basic syntax

```
Public Property ConnectionName As String
```

C# syntax

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

Exceptions

- ArgumentException The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

This is only needed if you create more than one connection to the database.

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

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

**Visual Basic syntax**

```
Public Property DatabaseOnDesktop As String
```

**C# syntax**

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

**Exceptions**

- `ArgumentException`  
  The value contained a semicolon, or began with either a single quote or a double quote.

**Remarks**

A string specifying the absolute or relative path to the database. If the value is a null reference (Nothing in Visual Basic), the database ulstore.udb is used. In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

---

**DatabaseOnDevice property**

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

**Visual Basic syntax**

```
Public Property DatabaseOnDevice As String
```

**C# syntax**

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

**Exceptions**

- `ArgumentException`  
  The value contained a semicolon, or began with either a single quote or a double quote.

**Remarks**

A string specifying the full path to the database. If the value is a null reference (Nothing in Visual Basic), the database \UltraLiteDB\ulstore.udb is used. In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

---

**EncryptionKey property**

Specifies a key for encrypting the database.

**Visual Basic syntax**

```
Public Property EncryptionKey As String
```
C# syntax

```csharp
public string EncryptionKey {get;set;}
```

Exceptions

- `ArgumentException`  The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

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

All connections must use the same key as was specified when the database was created. Lost or forgotten keys result in completely inaccessible databases.

As with all passwords, it is best to choose a key value that cannot be easily guessed. The key can be of arbitrary length, but generally the longer the key, the better, because a shorter key is easier to guess than a longer one. Using a combination of numbers, letters, and special characters decreases the chances of someone guessing the key.

See also

- “ULConnection.ChangeEncryptionKey method [UltraLite.NET]” on page 130

Password property

Specifies the password for the authenticated user.

Visual Basic syntax

```vbnet
Public Property Password As String
```

C# syntax

```csharp
public string Password {get;set;}
```

Exceptions

- `ArgumentException`  The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

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

Passwords are case sensitive.

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

See also

- “ULConnectionParms.UserID property [UltraLite.NET]” on page 172
**UserID property**

Specifies an authenticated user for the database.

**Visual Basic syntax**

```vbnet
Public Property UserID As String
```

**C# syntax**

```csharp
public string UserID {get;set;}
```

**Exceptions**

- **ArgumentException**  The value contained a semicolon, or began with either a single quote or a double quote.

**Remarks**

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

User IDs are case-insensitive.

Databases are initially created with a single authenticated user named DBA.

If both the user ID and password are not supplied, the user DBA with password sql are used. To make the database more secure, change the user DBA's password or create new users (with the ULConnection.GrantConnectTo method) and remove the DBA user (with the ULConnection.RevokeConnectFrom method).

**See also**

- “ULConnectionParms.Password property [UltraLite.NET]” on page 171
- “ULConnection.GrantConnectTo method [UltraLite.NET]” on page 145
- “ULConnection.RevokeConnectFrom method [UltraLite.NET]” on page 147

---

**ULConnectionStringBuilder class**

Builds a connection string for opening a connection to an UltraLite database.

**Visual Basic syntax**

```vbnet
Public NotInheritable Class ULConnectionStringBuilder
```

**C# syntax**

```csharp
```

**Base classes**

Members

All members of the ULConnectionStringBuilder class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULConnectionStringBuilder constructor</td>
<td>Initializes a ULConnectionStringBuilder object with its default values.</td>
</tr>
<tr>
<td>AppendKeyValuePair method (Inherited from System.Data.Common.DbConnectionStringBuilder)</td>
<td>Provides an efficient and safe way to append a key and value to an existing System.Text.StringBuilder object.</td>
</tr>
<tr>
<td>ContainsKey method</td>
<td>Determines whether the ULConnectionStringBuilder object contains a specific keyword.</td>
</tr>
<tr>
<td>EquivalentTo method</td>
<td>Compares the connection information in this ULConnectionStringBuilder object with the connection information in the supplied DbConnectionStringBuilder object.</td>
</tr>
<tr>
<td>GetShortName method</td>
<td>Retrieves the short version of the supplied keyword.</td>
</tr>
<tr>
<td>Remove method</td>
<td>Removes the entry with the specified key from the ULConnectionStringBuilder object.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TryGetValue method</td>
<td>Retrieves a value corresponding to the supplied key from this ULConnectionStringBuilder object.</td>
</tr>
<tr>
<td>CacheSize property</td>
<td><strong>UL Ext:</strong> Specifies the size of the cache.</td>
</tr>
<tr>
<td>ConnectionName property</td>
<td>Specifies a name for the connection.</td>
</tr>
<tr>
<td>DatabaseKey property</td>
<td>Specifies a key for encrypting the database.</td>
</tr>
<tr>
<td>DatabaseName property</td>
<td>Specifies a name for the database or the name of a loaded database to which a connection needs to be made.</td>
</tr>
<tr>
<td>DatabaseOnDesktop property</td>
<td><strong>UL Ext:</strong> Specifies the path and file name of the UltraLite database on Windows desktop platforms.</td>
</tr>
<tr>
<td>DatabaseOnDevice property</td>
<td><strong>UL Ext:</strong> Specifies the path and file name of the UltraLite database on Windows Mobile.</td>
</tr>
<tr>
<td>OrderedTableScans property</td>
<td>Specifies whether SQL queries without ORDER BY clauses should perform ordered table scans by default.</td>
</tr>
<tr>
<td>Password property</td>
<td>Specifies the password for the authenticated user.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ReserveSize property</td>
<td><strong>UL Ext:</strong> Specifies the reserve file system space for storage of UltraLite persistent data.</td>
</tr>
<tr>
<td>StartLine property</td>
<td>Specifies the location and then starts the UltraLite engine.</td>
</tr>
<tr>
<td>this property</td>
<td>Specifies the value of the specified connection keyword.</td>
</tr>
<tr>
<td>UserID property</td>
<td>Specifies an authenticated user for the database.</td>
</tr>
</tbody>
</table>

**Remarks**

The frequently-used connection parameters are individual properties on the ULConnectionStringBuilder object.

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

A ULConnectionStringBuilder object is used to specify the parameters for opening a connection (with the ULConnection.Open method) or dropping a database (with the ULDatabaseManager.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 ULConnectionStringBuilder object, you create a connection string using the `System.Data.Common.DbConnectionStringBuilder.ConnectionString`. The resulting string is used to create a new ULConnection object with the ULConnection(String) constructor or set the ULConnection.ConnectionString property of an existing ULConnection object.

**Identifying the database**

Each instance contains platform-specific paths to the database. Only the value corresponding to the executing platform is used. For example, in the code below the path `\UltraLite\mydb1.udb` would be used on Windows Mobile, while `mydb2.db` would be used on other platforms.

```
' Visual Basic
Dim dbName As ULConnectionStringBuilder = _
    New ULConnectionStringBuilder
dbName.DatabaseOnDevice = "\UltraLite\mydb1.udb"
dbName.DatabaseOnDesktop = "somedir\mydb2.udb"
```

The following code is the C# language equivalent:

```
// C#
ULConnectionStringBuilder dbName = new ULConnectionStringBuilder();
```
dbName.DatabaseOnDevice = "\\UltraLite\\mydb1.udb";
dbName.DatabaseOnDesktop = @"somedir\\mydb2.udb";

The recommended extension for UltraLite database files is .udb. On Windows Mobile devices, the default
database is \UltraLiteDB\ulstore.udb. On other Windows platforms, the default database is ulstore.udb. In
C#, you must escape any backslash characters in paths or use @-quoted string literals.

If you are using multiple databases, you must specify a database name for each database.

**Optional connection settings**

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

**See also**

- “ULConnection.Open method [UltraLite.NET]” on page 145
- “ULDatabaseManager.DropDatabase method [UltraLite.NET]” on page 216
- “ULConnection class [UltraLite.NET]” on page 118
- “ULConnectionStringBuilder.ConnectionString property [UltraLite.NET]” on page 154
- “ULConnectionStringBuilder.DatabaseName property [UltraLite.NET]” on page 181
- “ULConnectionStringBuilder.UserID property [UltraLite.NET]” on page 187
- “ULConnectionStringBuilder.Password property [UltraLite.NET]” on page 184
- “ULConnectionStringBuilder.DatabaseKey property [UltraLite.NET]” on page 181
- “ULConnectionStringBuilder.CacheSize property [UltraLite.NET]” on page 180
- “ULConnectionStringBuilder.ConnectionName property [UltraLite.NET]” on page 180
- “ULConnection.RevokeConnectFrom method [UltraLite.NET]” on page 147
- “ULConnection.GrantConnectTo method [UltraLite.NET]” on page 145
- “ULConnection.ChangeEncryptionKey method [UltraLite.NET]” on page 130
- “UltraLite connection parameters” [UltraLite - Database Management and Reference]
Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULConnectionStringBuilder() constructor</td>
<td>Initializes a ULConnectionStringBuilder object with its default values.</td>
</tr>
<tr>
<td>ULConnectionStringBuilder(string) constructor</td>
<td>Initializes a ULConnectionStringBuilder object with the specified connection string.</td>
</tr>
</tbody>
</table>

**ULConnectionStringBuilder() constructor**

Initializes a ULConnectionStringBuilder object with its default values.

**Visual Basic syntax**

```
Public Sub New()
```

**C# syntax**

```
public ULConnectionStringBuilder()
```

**ULConnectionStringBuilder(string) constructor**

Initializes a ULConnectionStringBuilder object with the specified connection string.

**Visual Basic syntax**

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

**C# syntax**

```
public ULConnectionStringBuilder(string connectionString)
```

**Parameters**

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

**ContainsKey method**

Determines whether the ULConnectionStringBuilder object contains a specific keyword.

**Visual Basic syntax**

```
Public Overrides Function ContainsKey(ByVal keyword As String) As Boolean
```

**C# syntax**

```
public override bool ContainsKey(string keyword)
```
Parameters
● **keyword**  The name of the connection keyword.

Returns
True if this connection string builder contains a value for the specified keyword, otherwise returns false.

**EquivalentTo method**
Compares the connection information in this ULConnectionStringBuilder object with the connection information in the supplied DbConnectionStringBuilder object.

**Visual Basic syntax**
Public Overrides Function EquivalentTo(ByVal connectionStringBuilder As DbConnectionStringBuilder) As Boolean

**C# syntax**
public override bool EquivalentTo(DbConnectionStringBuilder connectionStringBuilder)

Parameters
● **connectionStringBuilder**  The other DbConnectionStringBuilder object to compare this ULConnectionStringBuilder object to.

Returns
True if this object is equivalent to the specified DbConnectionStringBuilder object; otherwise, returns false.

See also

**GetShortName method**
Retrieves the short version of the supplied keyword.

**Visual Basic syntax**
Public Shared Function GetShortName(ByVal keyword As String) As String

**C# syntax**
public static string GetShortName(string keyword)

Parameters
● **keyword**  The key of the item to retrieve.
Returns
   The short version of the supplied keyword if keyword is recognized, null otherwise.

Remove method
   Removes the entry with the specified key from the ULConnectionStringBuilder object.

Visual Basic syntax
   Public Overrides Function Remove(ByVal keyword As String) As Boolean

C# syntax
   public override bool Remove(string keyword)

Parameters
   ● keyword   The name of the connection keyword.

Returns
   True if the key existed within the connection string and was removed; false if the key did not exist.

TryGetValue method
   Retrieves a value corresponding to the supplied key from this ULConnectionStringBuilder object.

Visual Basic syntax
   Public Overrides Function TryGetValue(ByVal keyword As String, ByVal value As Object) As Boolean

C# syntax
   public override bool TryGetValue(string keyword, out Object value)

Parameters
   ● keyword   The key of the item to retrieve.
   ● value     The value corresponding to the key.

Returns
   True if keyword was found within the connection string, false otherwise.

Remarks
   The TryGetValue method lets developers safely retrieve a value from a ULConnectionStringBuilder without needing to first call the ContainsKey method. Because the TryGetValue method does not raise an exception when you call it, passing in a nonexistent key, you do not have to look for a key before
retrieve its value. Calling TryGetValue with a nonexistent key places the null value (Nothing in Visual Basic) in the value parameter.

**CacheSize property**

**UL Ext:** Specifies the size of the cache.

**Visual Basic syntax**

```vbnet
Public Property CacheSize As String
```

**C# syntax**

```csharp
public string CacheSize {get;set;}
```

**Exceptions**

- **ArgumentException**  The value contained a semicolon, or began with either a single quote or a double quote.

**Remarks**

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

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

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

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

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

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

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

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

**ConnectionName property**

Specifies a name for the connection.

**Visual Basic syntax**

```vbnet
Public Property ConnectionName As String
```
C# syntax

```csharp
public string ConnectionName {get;set;}
```

Exceptions

- **ArgumentException**
  The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

This is only needed if you create more than one connection to the database.

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

**DatabaseKey property**

Specifies a key for encrypting the database.

Visual Basic syntax

```vbnet
Public Property DatabaseKey As String
```

C# syntax

```csharp
public string DatabaseKey {get;set;}
```

Exceptions

- **ArgumentException**
  The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

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

All connections must use the same key as was specified when the database was created. Lost or forgotten keys result in completely inaccessible databases.

As with all passwords, it is best to choose a key value that cannot be easily guessed. The key can be of arbitrary length, but generally the longer the key, the better, because a shorter key is easier to guess than a longer one. Using a combination of numbers, letters, and special characters decreases the chances of someone guessing the key.

See also

- “ULConnection.ChangeEncryptionKey method [UltraLite.NET]” on page 130

**DatabaseName property**

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

Public Property DatabaseName As String

C# syntax

public string DatabaseName {get;set;}

Exceptions

- ArgumentException
  The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

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

When a database is started, it is assigned a database name, either explicitly with the dbn parameter, or by UltraLite using the base of the file name with the extension and path removed.

When opening connections, UltraLite first searches for a running database with a matching dbn parameter. If one is not found, UltraLite starts a new database using the appropriate database file name parameter (the DatabaseOnDevice or DatabaseOnDesktop properties).

This parameter is required if the application (or UltraLite engine) needs to access two different databases that have the same base file name.

See also

- “ULConnectionStringBuilder.DatabaseOnDevice property [UltraLite.NET]” on page 183
- “ULConnectionStringBuilder.DatabaseOnDesktop property [UltraLite.NET]” on page 182

DatabaseOnDesktop property

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

Visual Basic syntax

Public Property DatabaseOnDesktop As String

C# syntax

public string DatabaseOnDesktop {get;set;}

Exceptions

- ArgumentException
  The value contained a semicolon, or began with either a single quote or a double quote.

Remarks

A string specifying the absolute or relative path to the database. If the value is a null reference (Nothing in Visual Basic), the database ulstore.udb is used. In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).
**DatabaseOnDevice property**

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

**Visual Basic syntax**

```vbnet
Public Property DatabaseOnDevice As String
```

**C# syntax**

```csharp
public string DatabaseOnDevice {get;set;}
```

**Exceptions**

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

**Remarks**

A string specifying the full path to the database. If the value is a null reference (Nothing in Visual Basic), the database `\UltraLiteDB\ulstore.udb` is used. In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

**OrderedTableScans property**

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

**Visual Basic syntax**

```vbnet
Public Property OrderedTableScans As String
```

**C# syntax**

```csharp
public string OrderedTableScans {get;set;}
```

**Exceptions**

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

**Remarks**

A boolean string specifying whether to use ordered table scans or not. For example, true/false, yes/no, 1/0, and so on. The default value is a null reference (Nothing in Visual Basic).

When using dynamic SQL in UltraLite, if order is not important for executing a query, UltraLite accesses the rows directly from the database pages rather than using the primary key index. This improves performance of fetching rows. To use this optimization, the query must be read only and must scan all the rows.

When rows are expected in a specific order, an ORDER BY statement should be included as part of the SQL query. However, it's possible that some applications have come to rely on the behavior that defaults
to returning rows in the primary key order. In this case, users should set the OrderedTableScans parameter to 1 (true, yes, on) to revert to the old behavior when iterating over a table.

When the OrderedTableScans value is set to 1 (true, yes, on) and the user does not specify an ORDER BY clause or if a query would not benefit from an index, UltraLite defaults to using the primary key.

**Password property**

Specifies the password for the authenticated user.

**Visual Basic syntax**

```vban
Public Property Password As String
```

**C# syntax**

```c#
public string Password {get;set;}
```

**Exceptions**

- **ArgumentException**  The value contained a semicolon, or began with either a single quote or a double quote.

**Remarks**

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

Passwords are case sensitive.

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

**See also**

- “ULConnectionStringBuilder.UserID property [UltraLite.NET]” on page 187

**ReserveSize property**

**UL Ext:** Specifies the reserve file system space for storage of UltraLite persistent data.

**Visual Basic syntax**

```vban
Public Property ReserveSize As String
```

**C# syntax**

```c#
public string ReserveSize {get;set;}
```

**Exceptions**

- **ArgumentException**  The value contained a semicolon, or began with either a single quote or a double quote.
Remarks
A string specifying the reserve size. The default is a null reference (Nothing in Visual Basic).

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

The reserve_size parameter allows you to pre-allocate the file system space required for your UltraLite database without inserting any data. Reserving file system space can improve performance slightly and also prevent out of memory failures. By default, the persistent storage file only grows when required as the application updates the database.

The reserve_size reserves file system space, which includes the metadata in the persistent store file, and not just the raw data. The metadata overhead and data compression must be considered when deriving the required file system space from the amount of database data.

The reserve_size parameter reserves space by growing the persistent store file to the given reserve size on startup, regardless of whether the file previously existed. The file is never truncated.

The following parameter string ensures that the persistent store file is at least 2 MB upon startup.

```
connParms.ReserveSize = "2m"
```

StartLine property
Specifies the location and then starts the UltraLite engine.

Visual Basic syntax
```
Public Property StartLine As String
```

C# syntax
```
public string StartLine {get;set;}
```

Exceptions
- ArgumentException The value contained a semicolon, or began with either a single quote or a double quote.

Remarks
A string specifying the location of the UltraLite engine executable. The default value is a null reference (Nothing in Visual Basic).

Only supply a StartLine (START) connection parameter if you are connecting to an engine that is not currently running.

See also
- “ULDatabaseManager.RuntimeType property [UltraLite.NET]” on page 220
**this property**

Specifies the value of the specified connection keyword.

**Visual Basic syntax**

```vbnet
Public Overrides Property Item(ByVal keyword As String) As Object
```

**C# syntax**

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

**Parameters**

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

**Remarks**

An object representing the value of the specified connection keyword.

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

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Corresponding Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache_size</td>
<td>ULConnectionStringBuilder.CacheSize</td>
</tr>
<tr>
<td>ce_file</td>
<td>ULConnectionStringBuilder.DatabaseOnDevice</td>
</tr>
<tr>
<td>con</td>
<td>ULConnectionStringBuilder.ConnectionName</td>
</tr>
<tr>
<td>dbkey</td>
<td>ULConnectionStringBuilder.DatabaseKey</td>
</tr>
<tr>
<td>dbn</td>
<td>ULConnectionStringBuilder.DatabaseName</td>
</tr>
<tr>
<td>nt_file</td>
<td>ULConnectionStringBuilder.DatabaseOnDesktop</td>
</tr>
<tr>
<td>pwd</td>
<td>ULConnectionStringBuilder.Password</td>
</tr>
<tr>
<td>reserve_size</td>
<td>ULConnectionStringBuilder.ReserveSize</td>
</tr>
<tr>
<td>start</td>
<td>ULConnectionStringBuilder.StartLine</td>
</tr>
<tr>
<td>uid</td>
<td>ULConnectionStringBuilder.UserID</td>
</tr>
</tbody>
</table>
See also

- “ULConnectionStringBuilder.CacheSize property [UltraLite.NET]” on page 180
- “ULConnectionStringBuilder.DatabaseOnDevice property [UltraLite.NET]” on page 183
- “ULConnectionStringBuilder.ConnectionName property [UltraLite.NET]” on page 180
- “ULConnectionStringBuilder.DatabaseKey property [UltraLite.NET]” on page 181
- “ULConnectionStringBuilder.DatabaseName property [UltraLite.NET]” on page 181
- “ULConnectionStringBuilder.DatabaseOnDesktop property [UltraLite.NET]” on page 182
- “ULConnectionStringBuilder.Password property [UltraLite.NET]” on page 184
- “ULConnectionStringBuilder.ReserveSize property [UltraLite.NET]” on page 184
- “ULConnectionStringBuilder.StartLine property [UltraLite.NET]” on page 185
- “ULConnectionStringBuilder.UserID property [UltraLite.NET]” on page 187

**UserID property**

Specifies an authenticated user for the database.

**Visual Basic syntax**

```vbnet
Public Property UserID As String
```

**C# syntax**

```csharp
public string UserID {get;set;}
```

**Exceptions**

- **ArgumentException** The value contained a semicolon, or began with either a single quote or a double quote.

**Remarks**

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

User IDs are case-insensitive.

Databases are initially created with a single authenticated user named DBA.

If both the user ID and password are not supplied, the user DBA with password sql are used. To make the database more secure, change the user DBA’s password or create new users (with the ULConnection.GrantConnectTo method) and remove the DBA user (with the ULConnection.RevokeConnectFrom method).

See also

- “ULConnectionStringBuilder.Password property [UltraLite.NET]” on page 184
- “ULConnection.GrantConnectTo method [UltraLite.NET]” on page 145
- “ULConnection.RevokeConnectFrom method [UltraLite.NET]” on page 147
ULCreateParms class

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

**Visual Basic syntax**

```vbnet
Public Class ULCreateParms
```

**C# syntax**

```csharp
public class ULCreateParms
```

**Members**

All members of the ULCreateParms class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULCreateParms constructor</td>
<td>Initializes a ULCreateParms object with its default values.</td>
</tr>
<tr>
<td>ToString method</td>
<td>Returns the string representation of this instance.</td>
</tr>
<tr>
<td>CaseSensitive property</td>
<td>Specifies whether the new database should be case sensitive when comparing string values.</td>
</tr>
<tr>
<td>ChecksumLevel property</td>
<td>Specifies the level of database page checksums enabled for the new database.</td>
</tr>
<tr>
<td>DateFormat property</td>
<td>Specifies the date format used for string conversions by the new database.</td>
</tr>
<tr>
<td>DateOrder property</td>
<td>Specifies the date order used for string conversions by the new database.</td>
</tr>
<tr>
<td>FIPS property</td>
<td>Specifies whether the new database should be using AES_FIPS encryption or AES encryption.</td>
</tr>
<tr>
<td>MaxHashSize property</td>
<td>Specifies the default maximum number of bytes to use for index hashing in the new database.</td>
</tr>
<tr>
<td>NearestCentury property</td>
<td>Specifies the nearest century used for string conversions by the new database.</td>
</tr>
<tr>
<td>Obfuscate property</td>
<td>Specifies whether the new database should use obfuscation to encrypt the database.</td>
</tr>
<tr>
<td>PageSize property</td>
<td>Specifies the page size, in bytes or kilobytes, of the new database.</td>
</tr>
<tr>
<td>Precision property</td>
<td>Specifies the floating-point precision used for string conversions by the new database.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Scale property</td>
<td>Specifies the minimum number of digits after the decimal point when an</td>
</tr>
<tr>
<td></td>
<td>arithmetic result is truncated to the maximum precision during string</td>
</tr>
<tr>
<td></td>
<td>conversions by the new database.</td>
</tr>
<tr>
<td>TimeFormat property</td>
<td>Specifies the time format used for string conversions by the new database.</td>
</tr>
<tr>
<td>TimestampFormat property</td>
<td>Specifies the timestamp format used for string conversions by the new</td>
</tr>
<tr>
<td></td>
<td>database.</td>
</tr>
<tr>
<td>TimestampIncrement</td>
<td>Specifies the minimum difference between two unique timestamps, in</td>
</tr>
<tr>
<td>property</td>
<td>microseconds (1,000,000th of a second).</td>
</tr>
<tr>
<td>UTF8Encoding property</td>
<td>Specifies whether the new database should be using the UTF8 character</td>
</tr>
<tr>
<td></td>
<td>set or the character set associated with the collation.</td>
</tr>
</tbody>
</table>

**Remarks**

A ULCreateParms object is used to specify the parameters for creating a database with the ULDatabaseManager.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 ULCreateParms.ToString method. The resulting string can then be used as the createParms parameter of the ULDatabaseManager.CreateDatabase method.

**See also**

- “ULDatabaseSchema.GetDatabaseProperty method [UltraLite.NET]” on page 222
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” on page 215
- “ULCreateParms.ToString method [UltraLite.NET]” on page 190
- “UltraLite connection parameters” [UltraLite - Database Management and Reference]

**Example**

The following code creates the database \UltraLite\MyDatabase.udb on a Windows Mobile device. The database is created case sensitive and with the UTF8 character set.

```vbscript
' Visual Basic
Dim createParms As ULCreateParms = New ULCreateParms
createParms.CaseSensitive = True
createParms.UTF8Encoding = True
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"

ULConnection.DatabaseManager.CreateDatabase( _
    openParms.ToString(), _
    createParms.ToString() _
)```
Dim conn As ULConnection = _
    New ULConnection( openParms.ToString() )
conn.Open()

The following code is the C# language equivalent:

    // C#
    ULCreateParms createParms = new ULCreateParms();
    createParms.CaseSensitive = true;
    createParms.UTF8Encoding = true;
    ULConnectionParms openParms = new ULConnectionParms();
    openParms.DatabaseOnDevice = ".udb";

    ULConnection.DatabaseManager.CreateDatabase(
        openParms.ToString(),
        createParms.ToString()
    );

    ULConnection conn = new ULConnection( openParms.ToString() );
    conn.Open();

**ULCreateParms constructor**

Initializes a ULCreateParms object with its default values.

**Visual Basic syntax**

    Public Sub New()

**C# syntax**

    public ULCreateParms()

**ToString method**

Returns the string representation of this instance.

**Visual Basic syntax**

    Public Overrides Function ToString() As String

**C# syntax**

    public override string ToString()

**Returns**

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

**CaseSensitive property**

Specifies whether the new database should be case sensitive when comparing string values.
Visual Basic syntax
    Public Property CaseSensitive As Boolean

C# syntax
    public bool CaseSensitive {get;set;}

Remarks
    True if the database should be case sensitive; false if the database should be case insensitive. The default is false.

This method only affects how string data is compared and sorted. Database identifiers such as table names, column names, index names, and connection user IDs are always case insensitive. Connection passwords and database encryption keys are always case sensitive.

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

Visual Basic syntax
    Public Property ChecksumLevel As Integer

C# syntax
    public int ChecksumLevel {get;set;}

Exceptions
    ● ArgumentException  The value is invalid.

Remarks
    An integer specifying the checksum level. Valid values are 0, 1, and 2. The default is 0.

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

Visual Basic syntax
    Public Property DateFormat As String

C# syntax
    public string DateFormat {get;set;}

Exceptions
    ● ArgumentException  The value contained a semicolon, or began with either a single quote or a double quote.
Remarks
A string specifying the date format. If the value is a null reference (Nothing in Visual Basic), the database uses "YYYY-MM-DD". In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

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

**Visual Basic syntax**
```vbnet
Public Property DateOrder As ULDateOrder
```

**C# syntax**
```csharp
public ULDateOrder DateOrder {get;set;}
```

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

See also
- “ULDateOrder enumeration [UltraLite.NET]” on page 442

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

**Visual Basic syntax**
```vbnet
Public Property FIPS As Boolean
```

**C# syntax**
```csharp
public bool FIPS {get;set;}
```

Remarks
True if the database should be encrypted using AES_FIPS, false if the database should be encrypted with AES. The default is false.

Encryption must be turned on by supplying a value for the EncryptionKey connection parameter when the new database is created. If FIPS is true and no encryption key is supplied, the ULDatabaseManager.CreateDatabase method fails with a missing encryption key error.

See also
- “ULConnectionParms.EncryptionKey property [UltraLite.NET]” on page 170
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” on page 215
MaxHashSize property

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

Visual Basic syntax

Public Property MaxHashSize As Integer

C# syntax

public int MaxHashSize {get;set;}

Exceptions

● ArgumentException  The value is invalid.

Remarks

An integer specifying the maximum hash size. The value must be in the range [0,32]. The default is 8.

NearestCentury property

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

Visual Basic syntax

Public Property NearestCentury As Integer

C# syntax

public int NearestCentury {get;set;}

Exceptions

● ArgumentException  The value is invalid.

Remarks

An integer specifying the nearest century. The value must be in the range [0,100]. The default is 50.

Obfuscate property

Specifies whether the new database should use obfuscation to encrypt the database.

Visual Basic syntax

Public Property Obfuscate As Boolean

C# syntax

public bool Obfuscate {get;set;}
Remarks
True if the database should be encrypted using obfuscation, false if the database should not be obfuscated. The default is false.

This option is ignored if FIPS encryption is turned on with the ULCreateParms.FIPS property. The encryption key is ignored if obfuscation is turned on and a value is supplied for the EncryptionKey connection parameter when the new database is created.

See also
- “ULCreateParms.FIPS property [UltraLite.NET]” on page 192

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

Visual Basic syntax
Public Property PageSize As Integer

C# syntax
public int PageSize {get;set;}

Exceptions
- ArgumentException The value is invalid.

Remarks
An integer specifying the page size in bytes. Valid values are 1024 (1K), 2048 (2K), 4096 (4K), 8192 (8K), 16384 (16K). The default is 4096.

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

Visual Basic syntax
Public Property Precision As Integer

C# syntax
public int Precision {get;set;}

Exceptions
- ArgumentException The value is invalid.

Remarks
An integer specifying the precision. The value must be in the range [1,127]. The default is 30.
See also
- “ULCreateParms.Scale property [UltraLite.NET]” on page 195

**Scale property**

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision during string conversions by the new database.

**Visual Basic syntax**

```vbnet
Public Property Scale As Integer
```

**C# syntax**

```csharp
public int Scale {get;set;}
```

**Exceptions**

- `ArgumentException`  The value is invalid.

**Remarks**

An integer specifying the scale. The value must be in the range [0,127]. The default is 6.

The Scale value must be less than or equal to the Precision value; otherwise, an error occurs when creating the database.

**TimeFormat property**

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

**Visual Basic syntax**

```vbnet
Public Property TimeFormat As String
```

**C# syntax**

```csharp
public string TimeFormat {get;set;}
```

**Exceptions**

- `ArgumentException`  The value contained a semicolon, or began with either a single quote or a double quote.

**Remarks**

A string specifying the time format. If the value is a null reference (Nothing in Visual Basic), the database uses "HH:NN:SS.SSS". In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).
**TimestampFormat property**

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

**Visual Basic syntax**

```vbnet
Public Property TimestampFormat As String
```

**C# syntax**

```csharp
public string TimestampFormat {get;set;}
```

**Exceptions**

- **ArgumentException**  The value contained a semicolon, or began with either a single quote or a double quote.

**Remarks**

A string specifying the timestamp format. If the value is a null reference (Nothing in Visual Basic), the database uses "YYYY-MM-DD HH:NN:SS.SSS". In C#, you must escape any backslash characters in paths or use @-quoted string literals. The default is a null reference (Nothing in Visual Basic).

**TimestampIncrement property**

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

**Visual Basic syntax**

```vbnet
Public Property TimestampIncrement As Integer
```

**C# syntax**

```csharp
public int TimestampIncrement {get;set;}
```

**Exceptions**

- **ArgumentException**  The value is invalid.

**Remarks**

An integer specifying the timestamp increment. The value must be in the range [1,60000000]. The default is 1.

**UTF8Encoding property**

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

**Visual Basic syntax**

```vbnet
Public Property UTF8Encoding As Boolean
```
C# syntax

```csharp
public bool UTF8Encoding {get;set;}
```

Remarks

True if the database should use the UTF8 character set, false if the database should use the character set associated with the collation. The default is false.

Choose to use the UTF8 character set 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.

**ULCursorSchema class**

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

Visual Basic syntax

```vbnet
Public MustInherit Class ULCursorSchema
```

C# syntax

```csharp
public abstract class ULCursorSchema
```

Derived classes

- “ULResultSetSchema class [UltraLite.NET]” on page 362
- “ULTableSchema class [UltraLite.NET]” on page 421

Members

All members of the ULCursorSchema class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetColumnID method</td>
<td>Returns the column ID of the named column.</td>
</tr>
<tr>
<td>GetColumnName method</td>
<td>Returns the name of the column identified by the specified column ID.</td>
</tr>
<tr>
<td>GetColumnPrecision method</td>
<td>Returns the precision of the column identified by the specified column ID if the column is a numeric column (the NUMERIC SQL type).</td>
</tr>
<tr>
<td>GetColumnScale method</td>
<td>Returns the scale of the column identified by the specified column ID if the column is a numeric column (the NUMERIC SQL type).</td>
</tr>
<tr>
<td>GetColumnSize method</td>
<td>Returns the size of the column identified by the specified column ID if the column is a sized column (the BINARY or CHAR SQL types).</td>
</tr>
<tr>
<td>GetColumnSQLName method</td>
<td>Returns the name of the column identified by the specified column ID.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetColumnULDbType method</td>
<td>Returns the UltraLite.NET data type of the column identified by the specified column ID.</td>
</tr>
<tr>
<td>GetSchemaTable method</td>
<td>Returns a System.Data.DataTable that describes the column schema of the ULDataReader object.</td>
</tr>
<tr>
<td>ColumnCount property</td>
<td>Returns the number of columns in the cursor.</td>
</tr>
<tr>
<td>IsOpen property</td>
<td>Checks whether the cursor schema is currently open.</td>
</tr>
<tr>
<td>Name property</td>
<td>Returns the name of the cursor.</td>
</tr>
</tbody>
</table>

**Remarks**

This class is an abstract base class of the ULTableSchema class and the ULResultSetSchema class.

**Note**

For users porting from the iAnywhere.UltraLite namespace, Column IDs are 0-based, not 1-based as they are in the iAnywhere.UltraLite namespace.

**See also**

- “ULTableSchema class [UltraLite.NET]” on page 421
- “ULResultSetSchema class [UltraLite.NET]” on page 362

**GetColumnID method**

Returns the column ID of the named column.

**Visual Basic syntax**

```vbnet
Public Function GetColumnID(ByVal name As String) As Short
```

**C# syntax**

```csharp
public short GetColumnID(string name)
```

**Parameters**

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

**Returns**

The column ID of the named column.

**Exceptions**

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

Column IDs range from 0 to ColumnCount-1, inclusive.

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.ColumnName property [UltraLite.NET]” on page 203

GetColumnName method

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

Visual Basic syntax

Public Function GetColumnName(ByVal columnID As Integer) As String

C# syntax

public string GetColumnName(int columnID)

Parameters

- columnID  The ID of the column. The value must be in the range [0,ColumnCount-1].

Returns

The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned.

Exceptions

- ULException class  A SQL error occurred.

Remarks

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.ColumnName property [UltraLite.NET]” on page 203
**GetColumnPrecision method**

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

**Visual Basic syntax**

```vbnet
Public Function GetColumnPrecision(ByVal columnID As Integer) As Integer
```

**C# syntax**

```csharp
public int GetColumnPrecision(int columnID)
```

**Parameters**

- **columnID**  The ID of the column. The value must be in the range [0,ColumnCount-1].

**Returns**

The precision of the specified numeric column.

**Exceptions**

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

**See also**

- “ULCursorSchema.GetColumnULDbType method [UltraLite.NET]” on page 202
- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

**GetColumnScale method**

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

**Visual Basic syntax**

```vbnet
Public Function GetColumnScale(ByVal columnID As Integer) As Integer
```

**C# syntax**

```csharp
public int GetColumnScale(int columnID)
```

**Parameters**

- **columnID**  The ID of the column. The value must be in the range [0,ColumnCount-1].

**Returns**

The scale of the specified numeric column.

**Exceptions**

- **ULException class**  A SQL error occurred.
See also

- “ULCursorSchema.GetColumnULDbType method [UltraLite.NET]” on page 202
- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

**GetColumnSize method**

Returns the size of the column identified by the specified column ID if the column is a sized column (the BINARY or CHAR SQL types).

**Visual Basic syntax**

```vbnet
Public Function GetColumnSize(ByVal columnID As Integer) As Integer
```

**C# syntax**

```csharp
public int GetColumnSize(int columnID)
```

**Parameters**

- **columnID**  The ID of the column. The value must be in the range [0,ColumnCount-1].

**Returns**

The size of the specified sized column.

**Exceptions**

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

See also

- “ULCursorSchema.GetColumnULDbType method [UltraLite.NET]” on page 202
- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

**GetColumnSQLName method**

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

**Visual Basic syntax**

```vbnet
Public Function GetColumnSQLName(ByVal columnID As Integer) As String
```

**C# syntax**

```csharp
public string GetColumnSQLName(int columnID)
```

**Parameters**

- **columnID**  The ID of the column. The value must be in the range [0,ColumnCount-1].

**Returns**

The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned.
Exceptions

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

Remarks

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 because the GetColumnSQLName method always returns just the name of the column (without the table name as a prefix) for non-aliased, non-computed columns. While this behavior more closely resembles the behavior of other ADO.NET providers, it is more likely to produce non-unique names.

Column IDs and count may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

See also

- “ULCursorSchema.ColumnName property [UltraLite.NET]” on page 203
- “ULCursorSchema.GetColumnName method [UltraLite.NET]” on page 199
- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

**GetColumnULDbType method**

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

**Visual Basic syntax**

```vbnet
Public Function GetColumnULDbType(ByVal columnID As Integer) As ULDbType
```

**C# syntax**

```csharp
public ULDbType GetColumnULDbType(int columnID)
```

**Parameters**

- **columnID**  The ID of the column. The value must be in the range [0,ColumnCount-1].

**Returns**

A ULDbType enumerated integer.

**Exceptions**

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

**See also**

- “ULCursorSchema.ColumnName property [UltraLite.NET]” on page 203
- “ULDbType enumeration [UltraLite.NET]” on page 442
GetSchemaTable method

Returns a System.Data.DataTable that describes the column schema of the ULDataReader object.

Visual Basic syntax

Public Function GetSchemaTable() As DataTable

C# syntax

public DataTable GetSchemaTable()

Returns

A System.Data.DataTable that describes the column schema.

See also

● “ULDataReader.GetSchemaTable method [UltraLite.NET]” on page 249
● “ULDataReader class [UltraLite.NET]” on page 228
● System.Data.DataTable

ColumnCount property

Returns the number of columns in the cursor.

Visual Basic syntax

Public ReadOnly Property ColumnCount As Short

C# syntax

public short ColumnCount {get;}

Remarks

The number of columns in the cursor or 0 if the cursor schema is closed.

Column IDs range from 0 to ColumnCount-1, inclusive.

Column IDs and count might change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

IsOpen property

Checks whether the cursor schema is currently open.

Visual Basic syntax

Public ReadOnly Property IsOpen As Boolean

C# syntax

public bool IsOpen {get;}

ULCursorSchema class

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 203
Remarks
True if the cursor schema is currently open; false if the cursor schema is closed.

Name property
Returns the name of the cursor.

Visual Basic syntax
Public ReadOnly Property Name As String

C# syntax
public abstract string Name {get;}

Remarks
The name of the cursor as a string.

ULDataAdapter class
Represents a set of commands and a database connection used to fill a System.Data.DataSet and to update a database.

Visual Basic syntax
Public NotInheritable Class ULDataAdapter
    Implements System.ICloneable

C# syntax
public sealed class ULDataAdapter :
    System.Data.Common.DbDataAdapter,
    System.ICloneable

Base classes
● System.ICloneable

Members
All members of the ULDataAdapter class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULDataAdapter constructor</td>
<td>Initializes a ULDataAdapter object.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetFillParameters method</td>
<td>Returns the parameters set by the user when executing a SELECT statement.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>adapter)</td>
<td></td>
</tr>
<tr>
<td>taAdapter)</td>
<td></td>
</tr>
<tr>
<td><strong>Update method</strong> (Inherited from System.Data.Common.DbDataAdapter)</td>
<td>Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified array of System.Data.DataRow objects.</td>
</tr>
<tr>
<td><strong>DeleteCommand property</strong></td>
<td>Specifies a ULCommand object that is executed against the database when the DbDataAdapter.Update method is called to delete rows in the database that correspond to deleted rows in the System.Data.DataSet.</td>
</tr>
<tr>
<td><strong>FillCommandBehavior property</strong> (Inherited from System.Data.Com-</td>
<td>Gets or sets the behavior of the command used to fill the data adapter.</td>
</tr>
<tr>
<td>mon(DbDataAdapter)</td>
<td></td>
</tr>
<tr>
<td><strong>InsertCommand property</strong></td>
<td>Specifies a ULCommand object that is executed against the database when the DbDataAdapter.Update method is called to insert rows in the database that correspond to inserted rows in the System.Data.DataSet.</td>
</tr>
<tr>
<td><strong>TableMappings property</strong></td>
<td>Returns a collection that provides the master mapping between a source table and a System.Data.DataTable</td>
</tr>
<tr>
<td><strong>UpdateBatchSize property</strong> (Inherited from System.Data.Common.DbDa-</td>
<td>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.</td>
</tr>
<tr>
<td>taAdapter)</td>
<td></td>
</tr>
<tr>
<td><strong>UpdateCommand property</strong></td>
<td>Specifies a ULCommand object that is executed against the database when the System.Data.Common.DbDataAdapter.Update method is called to update rows in the database that correspond to updated rows in the System.Data.DataSet.</td>
</tr>
</tbody>
</table>
Name | Description
--- | ---
RowUpdated event | Occurs during an update after a command is executed against the data source.
RowUpdating event | Occurs during an update before a command is executed against the data source.
DefaultSourceTableName field (Inherited from System.Data.Common.DbDataAdapter) | The default name used by the System.Data.CommonDataAdapter object for table mappings.

**Remarks**

The System.Data.DataSet provides a way to work with data offline; that is, away from your UltraLite database. The ULDataAdapter class provides methods to associate a System.Data.DataSet with a set of SQL statements.

Since UltraLite is a local database and MobiLink has conflict resolution, the use of the ULDataAdapter is limited. For most purposes, the ULDataReader or ULTable classes provide more efficient access to data.

**See also**
- “ULDataReader class [UltraLite.NET]” on page 228
- “ULTable class [UltraLite.NET]” on page 401
- System.Data.DataSet
- System.Data.IDbDataAdapter
- System.Data.IDataAdapter
- System.IDisposable

**ULDataAdapter constructor**

Initializes a ULDataAdapter object.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULDataAdapter() constructor</td>
<td>Initializes a ULDataAdapter object.</td>
</tr>
<tr>
<td>ULDataAdapter(string, string) constructor</td>
<td>Initializes a ULDataAdapter object with the specified SELECT statement and connection string.</td>
</tr>
<tr>
<td>ULDataAdapter(string, ULConnection) constructor</td>
<td>Initializes a ULDataAdapter object with the specified SELECT statement and connection.</td>
</tr>
</tbody>
</table>
**Name** | **Description**
---|---
ULDataAdapter(ULCommand) constructor | Initializes a ULDataAdapter object with the specified SELECT statement.

**ULDataAdapter() constructor**
Initializes a ULDataAdapter object.

**Visual Basic syntax**
```
Public Sub New()
```

**C# syntax**
```
public ULDataAdapter()
```

**See also**
- “ULDataAdapter.ULDataAdapter constructor [UltraLite.NET]" on page 207

**ULDataAdapter(string, string) constructor**
Initializes a ULDataAdapter object with the specified SELECT statement and connection string.

**Visual Basic syntax**
```
Public Sub New(
    ByVal selectCommandText As String,
    ByVal selectConnectionString As String
)
```

**C# syntax**
```
public ULDataAdapter(
    string selectCommandText,
    string selectConnectionString
)
```

**Parameters**
- **selectCommandText**  A SELECT statement to be used by the ULDataAdapter.SelectCommand method.
- **selectConnectionString**  A connection string for an UltraLite.NET database.

**See also**
- “ULDataAdapter.ULDataAdapter constructor [UltraLite.NET]” on page 207
- “ULDataAdapter.SelectCommand property [UltraLite.NET]” on page 211
**ULDataAdapter(string, ULConnection) constructor**

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

**Visual Basic syntax**

```vbnet
Public Sub New(
    ByVal selectCommandText As String,
    ByVal selectConnection As ULConnection
)
```

**C# syntax**

```csharp
public ULDataAdapter(
    string selectCommandText,
    ULConnection selectConnection
)
```

**Parameters**

- **selectCommandText**  A SELECT statement to be used by the ULDataAdapter.SelectCommand method of the ULDataAdapter object.
- **selectConnection**  A ULConnection object that defines a connection to a database.

**See also**

- “ULDataAdapter.ULDataAdapter constructor [UltraLite.NET]” on page 207
- “ULDataAdapter.SelectCommand property [UltraLite.NET]” on page 211
- “ULConnection class [UltraLite.NET]” on page 118

**ULDataAdapter(ULCommand) constructor**

Initializes a ULDataAdapter object with the specified SELECT statement.

**Visual Basic syntax**

```vbnet
Public Sub New(ByVal selectCommand As ULCommand)
```

**C# syntax**

```csharp
public ULDataAdapter(ULCommand selectCommand)
```

**Parameters**

- **selectCommand**  A ULCommand object that is used during

**See also**

- “ULDataAdapter.ULDataAdapter constructor [UltraLite.NET]” on page 207
- “ULCommand class [UltraLite.NET]” on page 71
- System.Data.DataSet
GetFillParameters method

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

Visual Basic syntax

Public Shadows Function GetFillParameters() As ULParameter()

C# syntax

public new ULParameter[] GetFillParameters()

Returns

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

Remarks

This is the strongly-typed version of the System.Data.Common.DbDataAdapter.GetFillParameters method.

See also

● “ULParameter class [UltraLite.NET]” on page 307

DeleteCommand property

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

Visual Basic syntax

Public Shadows Property DeleteCommand As ULCommand

C# syntax

public new ULCommand DeleteCommand {get;set;}

Remarks

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

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

InsertCommand property

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

Visual Basic syntax

Public Shadows Property InsertCommand As ULCommand

C# syntax

public new ULCommand InsertCommand {get;set;}

Remarks

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

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


SelectCommand property


Visual Basic syntax

Public Shadows Property SelectCommand As ULCommand
C# syntax

```csharp
public new ULCommand SelectCommand {get;set;}
```

Remarks

A ULCommand object that is executed to fill the System.Data.DataSet.

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

If the SelectCommand property does not return any rows, then no tables are added to the System.Data.DataSet, and no exception is raised. The SELECT statement can also be specified in the ULDataAdapter(ULCommand), ULDataAdapter(String,ULConnection), or ULDataAdapter(String,String) constructors.


See also

- “ULCommand class [UltraLite.NET]” on page 71
- “ULDataAdapter.ULDataAdapter constructor [UltraLite.NET]” on page 207
- System.Data.DataSet
- System.Data.IDbDataAdapter.SelectCommand

TableMappings property

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

Visual Basic syntax

```vbnet
Public ReadOnly Shadows Property TableMappings As DataTableMappingCollection
```

C# syntax

```csharp
public new DataTableMappingCollection TableMappings {get;}
```

Remarks

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

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

This is the strongly-typed version of the System.Data.IDataAdapter.TableMappings property.
See also

- System.Data.DataTable
- System.Data.Common.DataTableMapping
- System.Data.DataTable
- System.Data.Common.DataTableMappingCollection
- System.Data.DataSet
- System.Data.IDataAdapter.TableMappings

**UpdateCommand property**

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

**Visual Basic syntax**

```vbnet
Public Shadows Property UpdateCommand As ULCommand
```

**C# syntax**

```csharp
public new ULCommand UpdateCommand {get;set;}
```

**Remarks**

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

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

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


**See also**

- “ULCommand class [UltraLite.NET]” on page 71
- “ULCommand.UpdatedRowSource property [UltraLite.NET]” on page 106
- System.Data.DataSet
- System.Data.IDbDataAdapter.UpdateCommand

**RowUpdated event**

Occurs during an update after a command is executed against the data source.

**Visual Basic syntax**

```vbnet
Public Event RowUpdated As ULRowUpdatedEventHandler
```
C# syntax

```csharp
public event ULRowUpdatedEventHandler RowUpdated;
```

Remarks

When an attempt to update is made, the event fires.

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

See also

- “ULRowUpdatedEventHandler delegate [UltraLite.NET]” on page 438

**RowUpdating event**

Occurs during an update before a command is executed against the data source.

Visual Basic syntax

```vbnet
Public Event RowUpdating As ULRowUpdatingEventHandler
```

C# syntax

```csharp
public event ULRowUpdatingEventHandler RowUpdating;
```

Remarks

When an attempt to update is made, the event fires.

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

See also

- “ULRowUpdatedEventHandler delegate [UltraLite.NET]” on page 438

**ULDatabaseManager class**

**UL Ext:** Provides static methods for creating, deleting, and validating databases.

Visual Basic syntax

```vbnet
Public NotInheritable Class ULDatabaseManager
```

C# syntax

```csharp
public sealed class ULDatabaseManager
```

Members

All members of the ULDatabaseManager class, including all inherited members.
### ULDatabaseManager class

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDatabase method</td>
<td>Creates a new UltraLite database.</td>
</tr>
<tr>
<td>DropDatabase method</td>
<td>Deletes the specified database.</td>
</tr>
<tr>
<td>SetActiveSyncListener method</td>
<td>Specifies the listener object used to process ActiveSync calls from the MobiLink provider for ActiveSync.</td>
</tr>
<tr>
<td>SetServerSyncListener method</td>
<td>Specifies the listener object used to process the specified server synchronizaion message.</td>
</tr>
<tr>
<td>SignalSyncIsComplete method</td>
<td>Signals the MobiLink provider for ActiveSync that an application has completed synchronization.</td>
</tr>
<tr>
<td>ValidateDatabase method</td>
<td>Performs low level and index validation on a database.</td>
</tr>
<tr>
<td>RuntimeType property</td>
<td>Specifies the UltraLite.NET runtime type.</td>
</tr>
</tbody>
</table>

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

### Example
The following example selects the UltraLite Engine runtime and creates a connection:

```vbnet
' Visual Basic
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT
Dim conn As ULConnection = New ULConnection
' The RuntimeType is now locked
```

The following code is the C# language equivalent:

```csharp
// C#
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT;
ULConnection conn = new ULConnection();
// The RuntimeType is now locked
```

### CreateDatabase method
Creates a new UltraLite database.

### Visual Basic syntax

```vbnet
Public Shared Sub CreateDatabase(
    ByVal connString As String,
    ByVal createParms As String
)
```

### C# syntax

```csharp
public static void CreateDatabase(string connString, string createParms)
```
Parameters

- **connString**  The parameters for identifying a database in the form of a semicolon-separated list of keyword-value pairs.

- **createParms**  The parameters used to configure the new database in the form of a semicolon-separated list of keyword-value pairs.

Exceptions

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

See also

- “ULConnection.Open method [UltraLite.NET]” on page 145
- “ULConnectionParms class [UltraLite.NET]” on page 163
- “ULCreateParms class [UltraLite.NET]” on page 188

Example

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

```vbnet
' Visual Basic
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnDevice = \"\UltraLite\MyDatabase.udb\"
ULConnection.DatabaseManager.CreateDatabase( _
    openParms.ToString(), _
    "" _
)
Dim conn As ULConnection = _
    New ULConnection(openParms.ToString())
conn.Open()
```

The following code is the C# language equivalent:

```csharp
// C#
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnDevice = ".udb";
ULConnection.DatabaseManager.CreateDatabase(
    openParms.ToString(),
    ""
);
ULConnection conn = new ULConnection(openParms.ToString());
conn.Open();
```

### DropDatabase method

Deletes the specified database.

**Visual Basic syntax**

Public Shared Sub DropDatabase(ByVal connString As String)

**C# syntax**

    public static void DropDatabase(string connString)
Parameters

- **connString** The parameters for identifying a database in the form of a semicolon-separated list of keyword-value pairs.

Exceptions

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

Remarks

You cannot drop a database that has open connections.

See also

- “ULConnection.Open method [UltraLite.NET]” on page 145
- “ULConnectionParms class [UltraLite.NET]” on page 163

Example

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

```visualbasic
' Visual Basic
Dim connParms As ULConnectionParms = New ULConnectionParms
connParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"
ULConnection.DatabaseManager.DropDatabase( _
    connParms.ToString() _
)
```

The following code is the C# language equivalent:

```csharp
// C#
ULConnectionParms connParms = new ULConnectionParms();
connParms.DatabaseOnDevice = ".udb";
ULConnection.DatabaseManager.DropDatabase(
    connParms.ToString() );
ULConnection conn = new ULConnection( openParms.ToString() );
conn.Open();
```

**SetActiveSyncListener method**

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

**Visual Basic syntax**

```visualbasic
Public Shared Sub SetActiveSyncListener( _
    ByVal appClassName As String, _
    ByVal listener As ULActiveSyncListener
)
```

**C# syntax**

```csharp
public static void SetActiveSyncListener( _
    string appClassName, _
    ULActiveSyncListener listener
)
```
Parameters

- **appClassName**  The unique class name for the application. This is the class name used when the application is registered for use with ActiveSync.

- **listener**  The ULActiveSyncListener object. Use null (Nothing in Visual Basic) to remove the previous listener.

Exceptions

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

Remarks

The *appClassName* parameter is the unique identifier used to identify the application. The application can only use one *appClassName* value at a time. While a listener is registered with a particular *appClassName* value, calls to the SetServerSyncListener or SetActiveSyncListener methods with a different *appClassName* value fail.

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

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

Applications should remove all listeners prior to exiting.

See also

- “ULDatabaseManager.SetServerSyncListener method [UltraLite.NET]” on page 218
- “ULDatabaseManager.SetActiveSyncListener method [UltraLite.NET]” on page 217
- “ULActiveSyncListener interface [UltraLite.NET]” on page 44
- “ULActiveSyncListener.ActiveSyncInvoked method [UltraLite.NET]” on page 44

**SetServerSyncListener method**

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

**Visual Basic syntax**

```vbnet
Public Shared Sub SetServerSyncListener(  ByVal messageName As String,  ByVal appClassName As String,  ByVal listener As ULServerSyncListener)
"
```

**C# syntax**

```csharp
public static void SetServerSyncListener(  string messageName,  string appClassName,  ULServerSyncListener listener
"
```

218 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
Parameters

- **messageName**  The name of the message.
- **appClassName**  The unique class name for the application. This is a unique identifier used to identify the application.
- **listener**  The ULServerSyncListener object. Use null (Nothing in Visual Basic) to remove the previous listener.

Exceptions

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

Remarks

The `appClassName` parameter is the unique identifier used to identify the application. The application may only use one `appClassName` value at a time. While a listener is registered with a particular `appClassName` value, calls to the `SetServerSyncListener` or `SetActiveSyncListener` methods with a different `appClassName` value fail.

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

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

Applications should remove all listeners before exiting.

See also

- “ULDatabaseManager.SetServerSyncListener method [UltraLite.NET]” on page 218
- “ULDatabaseManager.SetActiveSyncListener method [UltraLite.NET]” on page 217
- “ULServerSyncListener.ServerSyncInvoked method [UltraLite.NET]” on page 372

**SignalSyncIsComplete method**

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

**Visual Basic syntax**

```
Public Shared Sub SignalSyncIsComplete()
```

**C# syntax**

```
public static void SignalSyncIsComplete()
```

See also

- “ULDatabaseManager.SignalSyncIsComplete method [UltraLite.NET]” on page 219
- “ULActiveSyncListener.ActiveSyncInvoked method [UltraLite.NET]” on page 44
ValidateDatabase method

Performs low level and index validation on a database.

Visual Basic syntax

```vbnet
Public Shared Sub ValidateDatabase(
    ByVal start_parms As String,
    ByVal how As ULDBValid
)
```

C# syntax

```csharp
public static void ValidateDatabase(string start_parms, ULDBValid how)
```

Parameters

- **start_parms** The parameters for identifying a database in the form of a semicolon-separated list of keyword-value pairs.
- **how** Describes how to validate the database.

Exceptions

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

See also

- “ULConnection.ValidateDatabase method [UltraLite.NET]” on page 153
- “ULDBValid enumeration [UltraLite.NET]” on page 441
- “ULConnectionParms class [UltraLite.NET]” on page 163

Example

The following code validates indexes for the database \UltraLite\MyDatabase.udb under Windows Mobile:

```vbnet
' Visual Basic
Dim openParms As ULConnectionParms = New ULConnectionParms
openParms.DatabaseOnDevice = "\UltraLite\MyDatabase.udb"
ULConnection.DatabaseManager.ValidateDatabase( _
    openParms.ToString(), iAnywhere.Data.UltraLite.ULVF_INDEX )
```

The following code is the C# language equivalent:

```csharp
// C#
ULConnectionParms openParms = new ULConnectionParms();
openParms.DatabaseOnDevice = ".udb";
ULConnection.DatabaseManager.ValidateDatabase( _
    openParms.ToString(), iAnywhere.Data.UltraLite.ULVF_INDEX );
```

RuntimeType property

Specifies the UltraLite.NET runtime type.
Visual Basic syntax

Public Shared Property RuntimeType As ULRuntimeType

C# syntax

public ULRuntimeType RuntimeType {get;set;}

Remarks

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

A ULRuntimeType value identifying the type of the unmanaged UltraLite.NET runtime.

See also

● “ULRuntimeType enumeration [UltraLite.NET]” on page 446

Example

The following example selects the UltraLite Engine runtime and creates a connection:

' Visual Basic
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT
Dim conn As ULConnection = new ULConnection
' The RuntimeType is now locked

The following code is the C# language equivalent:

// C#
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT;
ULConnection conn = new ULConnection();
// The RuntimeType is now locked

ULDatabaseSchema class

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

Visual Basic syntax

Public NotInheritable Class ULDatabaseSchema

C# syntax

public sealed class ULDatabaseSchema

Members

All members of the ULDatabaseSchema class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDatabaseProperty method</td>
<td>Returns the value of the specified database property.</td>
</tr>
<tr>
<td>GetPublicationName method</td>
<td>Returns the name of the publication identified by the specified publication ID.</td>
</tr>
</tbody>
</table>
#### Name | Description
---|---
GetTableName method | Returns the name of the table identified by the specified table ID.
SetDatabaseOption method | Sets the value for the specified database option.
IsCaseSensitive property | Checks whether the database is case sensitive.
IsOpen property | Determines whether the database schema is open.
PublicationCount property | Counts the number of publications in the database.
TableCount property | Counts the number of tables in the database.

**Remarks**

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

**See also**
- “ULDatabaseSchema class [UltraLite.NET]” on page 221
- “ULConnection.Schema property [UltraLite.NET]” on page 158

#### GetDatabaseProperty method

Returns the value of the specified database property.

**Visual Basic syntax**

```vbnet
Public Function GetDatabaseProperty(ByVal name As String) As String
```

**C# syntax**

```csharp
public string GetDatabaseProperty(string name)
```

**Parameters**

- **name**  The name of the database property whose value you want to obtain. Property names are case insensitive.

**Returns**

The value of the property as a string.

**Exceptions**

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

**Remarks**

Recognized properties are:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaseSensitive</td>
<td>The status of the case sensitivity feature. Returns ON if the database is case sensitive. Otherwise, it returns OFF. Database case sensitivity affects how indexes on tables and result sets are sorted. Case sensitivity does not affect how a connection's ULConnectionParms.UserID and ULConnectionParms.Password values are verified. User IDs are always case insensitive and passwords are always case sensitive.</td>
</tr>
<tr>
<td>CharSet</td>
<td>The character set of the database.</td>
</tr>
<tr>
<td>ChecksumLevel</td>
<td>The level of database page checksums enabled for the database.</td>
</tr>
<tr>
<td>Collation</td>
<td>The name of the database's collation sequence.</td>
</tr>
<tr>
<td>ConnCount</td>
<td>The number of connections to the database.</td>
</tr>
<tr>
<td>date_format</td>
<td>The date format used for string conversions by the database. This format is not necessarily the same as the one used by System.DateTime.</td>
</tr>
<tr>
<td>date_order</td>
<td>The date order used for string conversions by the database.</td>
</tr>
<tr>
<td>Encryption</td>
<td>The type of encryption applied to the database. Returns None, Simple, AES, or AES_FIPS.</td>
</tr>
<tr>
<td>File</td>
<td>The file name of the database.</td>
</tr>
<tr>
<td>global_database_id</td>
<td>The value of the global_database_id option used for global autoincrement columns.</td>
</tr>
<tr>
<td>isolation_level</td>
<td>The value of the isolation_level option used for controlling the degree to which the operations in one transaction are visible to the operations in other concurrent transactions. This value is set on a per connection basis.</td>
</tr>
<tr>
<td>MaxHashSize</td>
<td>The default maximum number of bytes to use for index hashing. This property can be set on a per-index basis.</td>
</tr>
<tr>
<td>ml_remote_id</td>
<td>The value of the ml_remote_id option used for identifying the database during synchronization.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the database (DBN).</td>
</tr>
<tr>
<td>nearest_century</td>
<td>The nearest century used for string conversions by the database.</td>
</tr>
<tr>
<td>PageSize</td>
<td>The page size of the database, in bytes.</td>
</tr>
<tr>
<td>precision</td>
<td>The floating-point precision used for string conversions by the database.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>scale</td>
<td>The minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION value during string conversions by the database.</td>
</tr>
<tr>
<td>time_format</td>
<td>The time format used for string conversions by the database. This format is not necessarily the same as the one used by System.TimeSpan.</td>
</tr>
<tr>
<td>timestamp_format</td>
<td>The timestamp format used for string conversions by the database. This format is not necessarily the same as the one used by System.DateTime.</td>
</tr>
<tr>
<td>timestamp_increment</td>
<td>The minimum difference between two unique timestamps, in microseconds (1,000,000th of a second).</td>
</tr>
</tbody>
</table>

See also
- “ULDatabaseSchema.SetDatabaseOption method [UltraLite.NET]” on page 225
- “ULConnectionParms.UserID property [UltraLite.NET]” on page 172
- “ULConnectionParms.Password property [UltraLite.NET]” on page 171
- System.TimeSpan
- System.DateTime

**GetPublicationName method**

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

**Visual Basic syntax**

```vbnet
Public Function GetPublicationName(ByVal pubID As Integer) As String
```

**C# syntax**

```csharp
public string GetPublicationName(int pubID)
```

**Parameters**
- **pubID**  The ID of the publication. The value must be in the range [1,PublicationCount].

**Returns**

The publication name as a string.

**Exceptions**
- **ULException class**  A SQL error occurred.
Remarks

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

See also

● “ULDatabaseSchema.PublicationCount property [UltraLite.NET]” on page 227

GetTableName method

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

Visual Basic syntax

Public Function GetTableName(ByVal tableID As Integer) As String

C# syntax

public string GetTableName(int tableID)

Parameters

● tableID The ID of the table. The value must be in range [1, TableCount].

Returns

The table name as a string.

Exceptions

● ULException class A SQL error occurred.

Remarks

Table IDs may change during a schema upgrade. To correctly identify a table, access it by name or refresh the cached IDs after a schema upgrade.

See also

● “ULDatabaseSchema.TableCount property [UltraLite.NET]” on page 228

SetDatabaseOption method

Sets the value for the specified database option.

Visual Basic syntax

Public Sub SetDatabaseOption(
    ByVal name As String,
    ByVal value As String
)
C# syntax

```csharp
public void SetDatabaseOption(string name, string value)
```

Parameters

- **name**  The name of the database option. Option names are case insensitive.
- **value**  The new value for the option.

Exceptions

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

Remarks

Setting a database option results in a commit being performed.

Using this method while a transaction is active may cause unpredictable results and is not recommended. The call changes the isolation level of the connection, but does not update the ULTransaction.IsolationLevel value.

Recognized options are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>global_database_id</td>
<td>The value used for global autoincrement columns. The value must be in the range [0, System.UInt32.MaxValue]. The default is the ULConnection.INVALID_DATABASE_ID value (used to indicate that the database ID has not been set for the current database).</td>
</tr>
<tr>
<td>isolation_level</td>
<td>The value used to control the degree to which the operations in one transaction are visible to the operations in other concurrent transactions. The value must be one of &quot;read_uncommitted&quot; or &quot;read_committed&quot;. The default is &quot;read_committed&quot;. Setting the isolation_level on a connection to &quot;read_uncommitted&quot; is equivalent to wrapping all operations on that connection with BeginTransaction(System.Data.IsolationLevel.ReadUncommitted) and Commit() calls. Similarly, &quot;read_committed&quot; is equivalent to System.Data.IsolationLevel.ReadCommitted. SetDatabaseOption() should not be used to set the current transaction's isolation level; use BeginTransaction(IsolationLevel) instead. UltraLite's definition of each isolation level is slightly different than ADO.NET's documentation of IsolationLevel. This value is set on a per connection basis.</td>
</tr>
<tr>
<td>ml_remote_id</td>
<td>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.</td>
</tr>
</tbody>
</table>

See also

- “ULDatabaseSchema.GetDatabaseProperty method [UltraLite.NET]” on page 222
- “ULConnection.INVALID_DATABASE_ID field [UltraLite.NET]” on page 163
- System.UInt32.MaxValue
- “Isolation levels” [UltraLite - Database Management and Reference]
**IsCaseSensitive property**
Checks whether the database is case sensitive.

**Visual Basic syntax**
```
Public ReadOnly Property IsCaseSensitive As Boolean
```

**C# syntax**
```
public bool IsCaseSensitive {get;}
```

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

**Remarks**
True if the database is case sensitive, and false if the database is case insensitive.

Database case sensitivity affects how indexes on tables and result sets are sorted. Case sensitivity also affects how the ULConnectionParms.UserID and ULConnectionParms.Password values are verified.

**See also**
- “ULDatabaseSchema.GetDatabaseProperty method [UltraLite.NET]” on page 222
- “ULConnectionParms.UserID property [UltraLite.NET]” on page 172
- “ULConnectionParms.Password property [UltraLite.NET]” on page 171

**IsOpen property**
Determines whether the database schema is open.

**Visual Basic syntax**
```
Public ReadOnly Property IsOpen As Boolean
```

**C# syntax**
```
public bool IsOpen {get;}
```

**Remarks**
True if this database schema is currently open, false if this database schema is currently closed.

A ULDatabaseSchema object is open only if the connection it is attached to is open.

**PublicationCount property**
Counts the number of publications in the database.

**Visual Basic syntax**
```
Public ReadOnly Property PublicationCount As Integer
```

## ULDatabaseSchema class

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 227
C# syntax

```csharp
public int PublicationCount {get;}
```

Remarks

The number of publications in the database.

Publication IDs range from 1 to the PublicationCount value, inclusively.

Note

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

See also

- “ULDatabaseSchema.GetPublicationName method [UltraLite.NET]” on page 224

**TableCount property**

Counts the number of tables in the database.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property TableCount As Integer
```

C# syntax

```csharp
public int TableCount {get;}
```

Remarks

The number of tables in the database.

Table IDs range from 1 to the TableCount value, inclusively.

Note

Table IDs and counts may change during a schema upgrade. To correctly identify a table, access it by name or refresh the cached IDs and counts after a schema upgrade.

**ULDataReader class**

Represents a read-only bi-directional cursor in an UltraLite database.

**Visual Basic syntax**

```vbnet
Public Class ULDataReader
    Inherits System.Data.Common.DbDataReader
    Implements System.ComponentModel.IListSource
```

---

228 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
C# syntax

```csharp
public class ULDataReader : System.Data.Common.DbDataReader,
                           System.ComponentModel.IListSource
```

Base classes

- System.Data.Common.DbDataReader
- System.ComponentModel.IListSource

Derived classes

- “ULResultSet class [UltraLite.NET]” on page 339

Members

All members of the ULDataReader class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close method</td>
<td>Closes the cursor.</td>
</tr>
<tr>
<td>Dispose method</td>
<td>Releases all resources used by the current instance of the System.Data.Common.DbDataReader class.</td>
</tr>
<tr>
<td>GetBoolean method</td>
<td>Returns the value for the specified column as a System.Boolean.</td>
</tr>
<tr>
<td>GetByte method</td>
<td>Returns the value for the specified column as an unsigned 8-bit value (System.Byte).</td>
</tr>
<tr>
<td>GetBytes method</td>
<td>UL Ext: Returns the value for the specified column as an array of System.Bytes values.</td>
</tr>
<tr>
<td>GetChar method</td>
<td>This method is not supported in UltraLite.NET.</td>
</tr>
<tr>
<td>GetChars method</td>
<td>Copies a subset of the value for the specified ULDbType.LongVarchar column, beginning at the specified offset, to the specified offset of the destination System.Char array.</td>
</tr>
<tr>
<td>GetDataTypeName method</td>
<td>Returns the name of the specified column’s provider data type.</td>
</tr>
<tr>
<td>GetDateTime method</td>
<td>Returns the value for the specified column as a System.DateTime type with millisecond accuracy.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>GetDecimal method</td>
<td>Returns the value for the specified column as a System.Decimal type.</td>
</tr>
<tr>
<td>GetDouble method</td>
<td>Returns the value for the specified column as a System.Double type.</td>
</tr>
<tr>
<td>GetEnumerable method</td>
<td>Returns an System.Collections.IEnumerator value that iterates through the ULDataReader object.</td>
</tr>
<tr>
<td>GetFieldType method</td>
<td>Returns the System.Type value most appropriate for the specified column.</td>
</tr>
<tr>
<td>GetField&lt;T&gt;Value method (Inherited from System.Data.Common.DbDataReader)</td>
<td>Synchronously gets the value of the specified column as a type.</td>
</tr>
<tr>
<td>GetFieldValueAsync method (Inherited from System.Data.Common.DbDataReader)</td>
<td>Asynchronously gets the value of the specified column as a type.</td>
</tr>
<tr>
<td>GetFloat method</td>
<td>Returns the value for the specified column as a System.Single type.</td>
</tr>
<tr>
<td>GetGuid method</td>
<td>Returns the value for the specified column as a UUID (System.Guid) type.</td>
</tr>
<tr>
<td>GetInt16 method</td>
<td>Returns the value for the specified column as a System.Int16 type.</td>
</tr>
<tr>
<td>GetInt32 method</td>
<td>Returns the value for the specified column as a System.Int32 type.</td>
</tr>
<tr>
<td>GetInt64 method</td>
<td>Returns the value for the specified column as a System.Int64 type.</td>
</tr>
<tr>
<td>GetName method</td>
<td>Returns the name of the specified column.</td>
</tr>
<tr>
<td>GetOrdinal method</td>
<td>Returns the column ID of the named column.</td>
</tr>
<tr>
<td>GetProviderSpecificFieldType method (Inherited from System.Data.Common.DbDataReader)</td>
<td>Returns the provider-specific field type of the specified column.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetRowCount method</td>
<td><strong>UL Ext:</strong> Returns the number of rows in the cursor, within threshold.</td>
</tr>
<tr>
<td>GetSchemaTable method</td>
<td>Returns a System.Data.DataTable value that describes the column metadata of the ULDataReader object.</td>
</tr>
<tr>
<td>GetString method</td>
<td>Returns the value for the specified column as a System.String type.</td>
</tr>
<tr>
<td>GetTimeSpan method</td>
<td>Returns the value for the specified column as a System.TimeSpan type with millisecond accuracy.</td>
</tr>
<tr>
<td>GetUInt16 method</td>
<td>Returns the value for the specified column as a System.UInt16 type.</td>
</tr>
<tr>
<td>GetUInt32 method</td>
<td>Returns the value for the specified column as a System.UInt32 type.</td>
</tr>
<tr>
<td>GetUInt64 method</td>
<td>Returns the value for the specified column as a System.UInt64 type.</td>
</tr>
<tr>
<td>GetValue method</td>
<td>Returns the value of the specified column in its native format.</td>
</tr>
<tr>
<td>GetValues method</td>
<td>Returns all the column values for the current row.</td>
</tr>
<tr>
<td>IsDBNull method</td>
<td>Checks whether the value from the specified column is NULL.</td>
</tr>
<tr>
<td>IsDBNullAsync method (Inherited from System.Data.Common.DbDataReader)</td>
<td>An asynchronous version of System.Data.Common.DbDataReader.IsDBNull(System.Int32), which gets a value that indicates whether the column contains non-existent or missing values.</td>
</tr>
<tr>
<td>MoveAfterLast method</td>
<td><strong>UL Ext:</strong> Positions the cursor to after the last row of the cursor.</td>
</tr>
<tr>
<td>MoveBeforeFirst method</td>
<td><strong>UL Ext:</strong> Positions the cursor to before the first row of the cursor.</td>
</tr>
<tr>
<td>MoveFirst method</td>
<td><strong>UL Ext:</strong> Positions the cursor to the first row of the cursor.</td>
</tr>
<tr>
<td>MoveLast method</td>
<td><strong>UL Ext:</strong> Positions the cursor to the last row of the cursor.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>MoveNext method</strong></td>
<td><strong>UL Ext:</strong> Positions the cursor to the next row or after the last row if the cursor was already on the last row.</td>
</tr>
<tr>
<td><strong>MovePrevious method</strong></td>
<td><strong>UL Ext:</strong> Positions the cursor to the previous row or before the first row.</td>
</tr>
<tr>
<td><strong>MoveRelative method</strong></td>
<td><strong>UL Ext:</strong> Positions the cursor relative to the current row.</td>
</tr>
<tr>
<td><strong>NextResult method</strong></td>
<td>Advances the ULDataReader object to the next result when reading the results of batch SQL statements.</td>
</tr>
<tr>
<td><strong>Read method</strong></td>
<td>Positions the cursor to the next row, or after the last row if the cursor was already on the last row.</td>
</tr>
<tr>
<td><strong>ReadAsync method</strong></td>
<td>An asynchronous version of System.Data.Common.DbDataReader.Read, which advances the reader to the next record in a result set.</td>
</tr>
<tr>
<td><strong>Depth property</strong></td>
<td>Returns the depth of nesting for the current row.</td>
</tr>
<tr>
<td><strong>FieldCount property</strong></td>
<td>Returns the number of columns in the cursor.</td>
</tr>
<tr>
<td><strong>HasRows property</strong></td>
<td>Checks whether the ULDataReader object has one or more rows.</td>
</tr>
<tr>
<td><strong>IsBOF property</strong></td>
<td><strong>UL Ext:</strong> Checks whether the current row position is before the first row.</td>
</tr>
<tr>
<td><strong>IsClosed property</strong></td>
<td>Checks whether the cursor is currently open.</td>
</tr>
<tr>
<td><strong>IsEOF property</strong></td>
<td><strong>UL Ext:</strong> Checks whether the current row position is after the last row.</td>
</tr>
<tr>
<td><strong>RecordsAffected property</strong></td>
<td>Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.</td>
</tr>
<tr>
<td><strong>RowCount property</strong></td>
<td><strong>UL Ext:</strong> Returns the number of rows in the cursor.</td>
</tr>
<tr>
<td><strong>Schema property</strong></td>
<td><strong>UL Ext:</strong> Holds the schema of this cursor.</td>
</tr>
<tr>
<td><strong>this property</strong></td>
<td>Returns the value of the specified column in its native format.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>

**Remarks**

Cursors are sets of rows from either a table or the result set from a query.

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

```visualbasic
' Visual Basic
Dim cmd As ULCommand = new ULCommand("'
"SELECT emp_id FROM employee", conn
')
Dim reader As ULDataReader = cmd.ExecuteReader()
```

The following code is the C# language equivalent:

```csharp
// C#
ULCommand cmd = new ULCommand("'
"SELECT emp_id FROM employee", conn
");
ULDataReader reader = cmd.ExecuteReader();
```

**UL Ext:** The ADO.NET standard only requires forward-only motion through the result set, but ULDataReader objects are bi-directional. ULDataReader's Move methods provide you with full flexibility when moving through results.

A ULDataReader object is a read-only result set. If you need a more flexible object to manipulate results, use the ULCommand.ExecuteResultSet method, the ULCommand.ExecuteTable method, or the ULDataAdapter class. The ULDataReader class 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 class a much faster response time.

**UL Ext:** All columns of a ULDataReader object may be retrieved using the GetString method.

**See also**

- “ULCommand class [UltraLite.NET]” on page 71
- “ULCommand.ExecuteResultSet method [UltraLite.NET]” on page 95
- “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
- “ULDataAdapter class [UltraLite.NET]” on page 204
- “ULDataReader.GetString method [UltraLite.NET]” on page 250
- System.Data.Common.DbDataReader
- System.Data.IDataReader
- System.Data.IDataRecord
- System.IDisposable
Close method
Closes the cursor.

Visual Basic syntax
Public Overrides Sub Close()

C# syntax
public override void Close()

Exceptions
- ULException class A SQL error occurred.

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

GetBoolean method
Returns the value for the specified column as a System.Boolean.

Visual Basic syntax
Public Overrides Function GetBoolean(ByVal colID As Integer) As Boolean

C# syntax
public override bool GetBoolean(int colID)

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

Returns
The column value as a System.Boolean.

Exceptions
- ULException class A SQL error occurred.

See also
- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Boolean
**GetByte method**

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

**Visual Basic syntax**

```vbnet
Public Overrides Function GetByte(ByVal colID As Integer) As Byte
```

**C# syntax**

```csharp
public override byte GetByte(int colID)
```

**Parameters**

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

**Returns**

The column value as a System.Byte.

**Exceptions**

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

**See also**

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Byte

**GetBytes method**

UL Ext: Returns the value for the specified column as an array of System.Bytes values.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetBytes(int) method</td>
<td>UL Ext: Returns the value for the specified column as an array of System.Bytes values.</td>
</tr>
<tr>
<td>GetBytes(int, long, byte[], int, int) method</td>
<td>Copies a subset of the value for the specified ULDbType.LongBinary column, beginning at the specified offset, to the specified offset of the destination System.Byte array.</td>
</tr>
</tbody>
</table>

**GetBytes(int) method**

UL Ext: Returns the value for the specified column as an array of System.Bytes values.
Visual Basic syntax

Public Function GetBytes(ByVal colID As Integer) As Byte()

C# syntax

public byte[] GetBytes(int colID)

Parameters

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

Returns

The column value as an array of System.Bytes type.

Exceptions

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

Remarks

Only valid for columns of the ULDbType.Binary, ULDbType.LongBinary, or ULDbType.UniqueIdentifier types.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.GetBytes method [UltraLite.NET]” on page 235
- System.Byte

GetBytes(int, long, byte[], int, int) method

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

Visual Basic syntax

Public Overrides Function GetBytes(  
    ByVal colID As Integer,  
    ByVal srcOffset As Long,  
    ByVal dst As Byte(),  
    ByVal dstOffset As Integer,  
    ByVal count As Integer  
) As Long

C# syntax

public override long GetBytes(  
    int colID,  
    long srcOffset,  
    byte[] dst,  
    int dstOffset,

236 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
int count
)

Parameters

- **colID**  The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **srcOffset**  The start position in the column value. Zero is the beginning of the value.
- **dst**  The destination array.
- **dstOffset**  The start position in the destination array.
- **count**  The number of bytes to be copied.

Returns

The actual number of bytes copied.

Exceptions

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

Remarks

If you pass a *dst* buffer that is a null reference (Nothing in Visual Basic), the GetBytes method returns the length of the field in bytes.

The bytes at the *srcOffset* through *srcOffset +count -1* positions of the value are copied into the *dstOffset* through *dstOffset +count -1* positions, 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 are true, a ULException object with code ULSQLCode.SQLE_INVALID_PARAMETER is thrown and the destination is not modified:

- *srcOffset* is negative.
- *dstOffset* is negative.
- *count* is negative.
- *dstOffset +count* is greater than the *dst* length.

For other errors, a ULException object with the appropriate error code is thrown.
GetChar method

This method is not supported in UltraLite.NET.

Visual Basic syntax

Public Overrides Function GetChar(ByVal colID As Integer) As Char

C# syntax

public override char GetChar(int colID)

Parameters

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

Returns

This method is not supported in UltraLite.NET.

Exceptions

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

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.GetBytes method [UltraLite.NET]” on page 235
- “ULException class [UltraLite.NET]” on page 265
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Byte

GetChars method

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

Visual Basic syntax

Public Overrides Function GetChars(
    ByVal colID As Integer,
    ByVal srcOffset As Long,
    ByVal dst As Char(),
    ByVal dstOffset As Integer,
) As Char

UltraLite.NET API reference

238 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
C# syntax

```csharp
public override long GetChars(
    int colID,
    long srcOffset,
    char[] dst,
    int dstOffset,
    int count
) {
    // Implementation
}
```

Parameters

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

- **srcOffset** The start position in the column value. Zero is the beginning of the value.

- **dst** The destination array.

- **dstOffset** The start position in the destination array.

- **count** The number of characters to be copied.

Returns

The actual number of characters copied.

Exceptions

- **ULEException class** A SQL error occurred.

Remarks

If you pass a `dst` buffer that is a null reference (Nothing in Visual Basic), the GetChars method returns the length of the field in characters.

The characters at the `srcOffset` through `srcOffset + count - 1` positions of the value are copied into the `dstOffset` through `dstOffset + count - 1` positions, 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 instances are true, a ULEException object with code ULSQLCode.SQLE_INVALID_PARAMETER constant is thrown and the destination is not modified:

- The `srcOffset` value is negative.

- The `dstOffset` value is negative.

- The `count` value is negative.

- The `dstOffset + count` value is greater than the `dst` length.

For other errors, a ULEException object with the appropriate error code is thrown.
See also
- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULException class [UltraLite.NET]” on page 265
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Char

**GetDataTypeName method**

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

**Visual Basic syntax**

```vbnet
Public Overrides Function GetDataTypeName(ByVal colID As Integer) As String
```

**C# syntax**

```csharp
public override string GetDataTypeName(int colID)
```

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

**Returns**
A string corresponding to the column's ULDbType type.

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

See also
- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULCursorSchema.GetColumnULDbType method [UltraLite.NET]” on page 202
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- “ULDbType enumeration [UltraLite.NET]” on page 442

**GetDateTime method**

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

**Visual Basic syntax**

```vbnet
Public Overrides Function GetDateTime(ByVal colID As Integer) As Date
```

**C# syntax**

```csharp
public override DateTime GetDateTime(int colID)
```
Parameters

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

Returns

The column value as a System.DateTime type.

Exceptions

- **NullException**  A SQL error occurred.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.DateTime

GetDecimal method

Returns the value for the specified column as a System.Decimal type.

Visual Basic syntax

Public Overrides Function GetDecimal(ByVal colID As Integer) As Decimal

C# syntax

public override decimal GetDecimal(int colID)

Parameters

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

Returns

The column value as a System.Decimal type.

Exceptions

- **NullException**  A SQL error occurred.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Decimal
**GetDouble method**

Returns the value for the specified column as a System.Double type.

**Visual Basic syntax**

```
Public Overrides Function GetDouble(ByVal colID As Integer) As Double
```

**C# syntax**

```
public override double GetDouble(int colID)
```

**Parameters**

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

**Returns**

The column value as a System.Double type.

**Exceptions**

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

**See also**

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Double

**GetEnumerator method**

Returns an System.Collections.IEnumerator value that iterates through the ULDataReader object.

**Visual Basic syntax**

```
Public Overrides Function GetEnumerator() As System.Collections.IEnumerator
```

**C# syntax**

```
public override IEnumerator GetEnumerator()
```

**Returns**

A System.Collections.IEnumerator for the ULDataReader object.

**See also**

- System.Collections.IEnumerator
GetFieldType method

Returns the System.Type value most appropriate for the specified column.

Visual Basic syntax

Public Overrides Function GetFieldType(ByVal colID As Integer) As Type

C# syntax

public override Type GetFieldType(int colID)

Parameters

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

Returns

A System.Type value for the column.

Exceptions

- ULException class A SQL error occurred.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetDataTypeName method [UltraLite.NET]” on page 240
- “ULCursorSchema.GetColumnULDbType method [UltraLite.NET]” on page 202
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Type

GetFloat method

Returns the value for the specified column as a System.Single type.

Visual Basic syntax

Public Overrides Function GetFloat(ByVal colID As Integer) As Single

C# syntax

public override float GetFloat(int colID)

Parameters

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

Returns

The column value as a System.Single type.
GetGuid method

Returns the value for the specified column as a UUID (System.Guid) type.

Visual Basic syntax

Public Overrides Function GetGuid(ByVal colID As Integer) As Guid

C# syntax

public override Guid GetGuid(int colID)

Parameters

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

Returns

The column value as a GUID type.

Exceptions

- ULException class A SQL error occurred.

Remarks

This method is only valid for columns of the ULDbType.UniqueIdentifier type or the ULDbType.Binary type with length 16.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Single

GetInt16 method

Returns the value for the specified column as a System.Int16 type.
Visual Basic syntax

Public Overrides Function GetInt16(ByVal colID As Integer) As Short

C# syntax

public override short GetInt16(int colID)

Parameters

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

Returns

The column value as an System.Int16 type.

Exceptions

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

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Int16

**GetInt32 method**

Returns the value for the specified column as a System.Int32 type.

Visual Basic syntax

Public Overrides Function GetInt32(ByVal colID As Integer) As Integer

C# syntax

public override int GetInt32(int colID)

Parameters

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

Returns

The column value as a System.Int32 type.

Exceptions

- **ULException class**  A SQL error occurred.
See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Int32

GetInt64 method

Returns the value for the specified column as a System.Int64 type.

Visual Basic syntax

```vbnet
Public Overrides Function GetInt64(ByVal colID As Integer) As Long
```

C# syntax

```csharp
public override long GetInt64(int colID)
```

Parameters

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

Returns

The column value as an System.Int64 type.

Exceptions

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

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Int64

GetName method

Returns the name of the specified column.

Visual Basic syntax

```vbnet
Public Overrides Function GetName(ByVal colID As Integer) As String
```

C# syntax

```csharp
public override string GetName(int colID)
```

UlraLite.NET API reference

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
Parameters

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

Returns

The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned.

Exceptions

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

Remarks

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, the MyTable.ID value is the name of the only column in the result set for the "SELECT ID FROM MyTable" query.

This method is identical to the ULCursorSchema.GetColumnName method.

See also

- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- “ULDataReader.GetSchemaTable method [UltraLite.NET]” on page 249
- “ULCursorSchema.GetColumnName method [UltraLite.NET]” on page 199
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260

GetOrdinal method

Returns the column ID of the named column.

Visual Basic syntax

```vbnet
Public Overrides Function GetOrdinal( ByVal columnName As String ) As Integer
```

C# syntax

```csharp
public override int GetOrdinal(string columnName)
```

Parameters

- columnName  The name of the column.

Returns

The column ID of the named column.

Exceptions

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

Column IDs range from 0 to ULDatasReader.FieldCount-1, inclusively.

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, the MyTable.ID value is the name of the only column in the result set for the "SELECT ID FROM MyTable" query.

Column IDs and counts may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

This method is identical to the ULCursorSchema.GetColumnID method.

See also

- “ULDatasReader.GetSchemaTable method [UltraLite.NET]” on page 249
- “ULDatasReader.FieldCount property [UltraLite.NET]” on page 260
- “ULCursorSchema.GetColumnID method [UltraLite.NET]” on page 198

GetRowCount method

UL Ext: Returns the number of rows in the cursor, within threshold.

Visual Basic syntax

Public Function GetRowCount(ByVal threshold As Integer) As Integer

C# syntax

public int GetRowCount(int threshold)

Parameters

- threshold Threshold limit for row count.

Exceptions

- ULException class A SQL error occurred.

Remarks

The number of rows in the cursor.

The RowCount property is expensive with complex queries, as it requires passing through the cursor rows. By using the GetRowCount( threshold ) method, the caller can determine if there are at least threshold rows. If the number of rows is below the threshold, that number is returned; otherwise, threshold is returned. This can be called again with a higher threshold.

If threshold is 0, returns the RowCount property.

See also

- “ULDatasReader.RowCount property [UltraLite.NET]” on page 262
GetSchemaTable method

Returns a System.Data.DataTable value that describes the column metadata of the ULDataReader object.

Visual Basic syntax

Public Overrides Function GetSchemaTable() As DataTable

C# syntax

public override DataTable GetSchemaTable()

Returns

A System.Data.DataTable describing the schema of each column in the ULDataReader.

Remarks

The GetSchemaTable method returns metadata about each column in the following order:

<table>
<thead>
<tr>
<th><strong>DataTable column</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnName</td>
<td>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. In result sets, not all columns have names and not all column names are unique.</td>
</tr>
<tr>
<td>ColumnOrdinal</td>
<td>The ID of the column. The value is in the range [0,FieldCount-1].</td>
</tr>
<tr>
<td>ColumnSize</td>
<td>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.</td>
</tr>
<tr>
<td>NumericPrecision</td>
<td>The precision of a numeric column (ProviderType ULDbType.Decimal or ULDbType.Numeric) or DBNull if the column is not numeric.</td>
</tr>
<tr>
<td>NumericScale</td>
<td>The scale of a numeric column (ProviderType ULDbType.Decimal or ULDbType.Numeric) or DBNull if the column is not numeric.</td>
</tr>
<tr>
<td>IsUnique</td>
<td>True if the column is a non-computed unique column in the table (BaseTableName) it is taken from.</td>
</tr>
<tr>
<td>IsKey</td>
<td>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 the IsKey value set to true does not need to be the minimal set that uniquely identifies a row in the result set.</td>
</tr>
<tr>
<td>BaseCatalogName</td>
<td>The name of the catalog in the database that contains the column. For UltraLite.NET, this value is always DBNull.</td>
</tr>
<tr>
<td>BaseColumnName</td>
<td>The original name of the column in the BaseTableName table of the database, or DBNull if the column is computed or if this information cannot be determined.</td>
</tr>
</tbody>
</table>
**DataTable column** | **Description**
--- | ---
BaseSchema-Name | The name of the schema in the database that contains the column. For UltraLite.NET, this value is always DBNull.
BaseTableName | The name of the table in the database that contains the column, or DBNull if column is computed or if this information cannot be determined.
DataType | The .NET data type that is most appropriate for this type of column.
AllowDBNull | True if the column is nullable, false if the column is not nullable or if this information cannot be determined.
ProviderType | The ULDbType value of the column.
IsIdentity | True if the column is an identity column, false if it is not an identity column. For UltraLite.NET, this value is always false.
IsAutoIncrement | True if the column is an autoincrement or global autoincrement column, false otherwise (or if this information cannot be determined).
IsRowVersion | True if the column contains a persistent row identifier that cannot be written to, and has no meaningful value except to identity the row. For UltraLite.NET, this value is always false.
IsLong | True if the column is a ULDbType.LongVarchar or a ULDbType.LongBinary column, false otherwise.
IsReadOnly | True if the column is read-only, false if the column is modifiable or if its access cannot be determined.
IsAliased | True if the column name is an alias, false if it is not an alias.
IsExpression | True if the column is an expression, false if it is a column value.

See also
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- “ULDbType enumeration [UltraLite.NET]” on page 442
- System.Data.DataTable

**GetString method**

Returns the value for the specified column as a System.String type.
Visual Basic syntax
Public Overrides Function GetString(ByVal colID As Integer) As String

C# syntax
public override String GetString(int colID)

Parameters
● colID  The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns
The column value as a System.String type.

Exceptions
● ULEException class  A SQL error occurred.

See also
● “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
● “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
● “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
● System.String

GetTimeSpan method
Returns the value for the specified column as a System.TimeSpan type with millisecond accuracy.

Visual Basic syntax
Public Function GetTimeSpan(ByVal colID As Integer) As TimeSpan

C# syntax
public TimeSpan GetTimeSpan(int colID)

Parameters
● colID  The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.

Returns
The column value as a System.TimeSpan type.

Exceptions
● ULEException class  A SQL error occurred.
GetUInt16 method

Returns the value for the specified column as a System.UInt16 type.

Visual Basic syntax

Public Function GetUInt16(ByVal colID As Integer) As UShort

C# syntax

public ushort GetUInt16(int colID)

Parameters

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

Returns

The column value as an System.UInt16 type.

Exceptions

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

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.TimeSpan

GetUInt32 method

Returns the value for the specified column as a System.UInt32 type.

Visual Basic syntax

Public Function GetUInt32(ByVal colID As Integer) As UInteger

C# syntax

public uint GetUInt32(int colID)
Parameters

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

Returns

The column value as a System.UInt32 type.

Exceptions

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

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.UInt32

### GetUInt64 method

Returns the value for the specified column as a System.UInt64 type.

**Visual Basic syntax**

```vbnet
Public Function GetUInt64(ByVal colID As Integer) As ULong
```

**C# syntax**

```csharp
public ulong GetUInt64 (int colID)
```

Parameters

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

Returns

The column value as a System.UInt64 type.

Exceptions

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

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.UInt64
**GetValue method**

Returns the value of the specified column in its native format.

**Visual Basic syntax**

```
Public Overrides Function GetValue(ByVal colID As Integer) As Object
```

**C# syntax**

```
public override object GetValue(int colID)
```

**Parameters**

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

**Returns**

The column value as the .NET type most appropriate for the column or the DBNull type if column is NULL.

**Exceptions**

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

**Remarks**

This method is identical to using the ULDataReader.this[int] method.

**See also**

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260

---

**GetValues method**

Returns all the column values for the current row.

**Visual Basic syntax**

```
Public Overrides Function GetValues(ByVal values As Object()) As Integer
```

**C# syntax**

```
public override int GetValues(object[] values)
```

**Parameters**

- **values**  
The array of System.Objects to hold the entire row.
Returns
The number of column values retrieved. If the length of the array is greater than the number of columns (ULDataReader.FieldCount), only FieldCount items are retrieved and the rest of the array is left unchanged.

Exceptions
- ArgumentNullException The values array is NULL or has zero length.
- ULEException class A SQL error occurred.

Remarks
For most applications, the GetValues method provides an efficient means for retrieving all columns, rather than retrieving each column individually.

You can pass an System.Object array that contains fewer than the number of columns contained in the resulting row. Only the amount of data the System.Object array holds is copied to the array. You can also pass an System.Object array whose length is more than the number of columns contained in the resulting row.

This method returns the DBNull type for NULL database columns. For other columns, it returns the value of the column in its native format.

See also
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.GetValue method [UltraLite.NET]” on page 254
- System.Object

IsDBNull method
Checks whether the value from the specified column is NULL.

Visual Basic syntax
Public Overrides Function IsDBNull(ByVal colID As Integer) As Boolean

C# syntax
public override bool IsDBNull(int colID)

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

Returns
True if value is NULL, false if value is not NULL.
Exceptions

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

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260

**MoveAfterLast method**

**UL Ext:** Positions the cursor to after the last row of the cursor.

**Visual Basic syntax**

```
Public Sub MoveAfterLast()
```

**C# syntax**

```
public void MoveAfterLast()
```

Exceptions

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

**MoveBeforeFirst method**

**UL Ext:** Positions the cursor to before the first row of the cursor.

**Visual Basic syntax**

```
Public Sub MoveBeforeFirst()
```

**C# syntax**

```
public void MoveBeforeFirst()
```

Exceptions

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

**MoveFirst method**

**UL Ext:** Positions the cursor to the first row of the cursor.

**Visual Basic syntax**

```
Public Function MoveFirst() As Boolean
```

**C# syntax**

```
public bool MoveFirst()
```
Returns

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

Exceptions

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

**MoveLast method**

UL Ext: Positions the cursor to the last row of the cursor.

Visual Basic syntax

Public Function MoveLast() As Boolean

C# syntax

public bool MoveLast()

Returns

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

Exceptions

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

**MoveNext method**

UL Ext: Positions the cursor to the next row or after the last row if the cursor was already on the last row.

Visual Basic syntax

Public Function MoveNext() As Boolean

C# syntax

public bool MoveNext()

Returns

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

Exceptions

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

Remarks

This method is identical to the ULDataReader.Read method.

See also

- “ULDataReader.Read method [UltraLite.NET]” on page 259
MovePrevious method

UL Ext: Positions the cursor to the previous row or before the first row.

Visual Basic syntax

Public Function MovePrevious() As Boolean

C# syntax

public bool MovePrevious()

Returns

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

Exceptions

- ULException class A SQL error occurred.

MoveRelative method

UL Ext: Positions the cursor relative to the current row.

Visual Basic syntax

Public Function MoveRelative(ByVal offset As Integer) As Boolean

C# syntax

public bool MoveRelative(int offset)

Parameters

- offset The number of rows to move. Negative values correspond to moving backward.

Returns

True if successful, false otherwise. For example, the method fails if it positions beyond the first or last row.

Exceptions

- ULException class A SQL error occurred.

Remarks

If the row does not exist, the method returns false, and the cursor position is after the last row (the ULDataReader.IsEOF method) if offset value is positive, and before the first row (the ULDataReader.IsBOF method) if the offset value is negative.

See also

- “ULDataReader.IsEOF property [UltraLite.NET]” on page 261
- “ULDataReader.IsBOF property [UltraLite.NET]” on page 261
**NextResult method**

Advances the ULDataReader object to the next result when reading the results of batch SQL statements.

**Visual Basic syntax**

```
Public Overrides Function NextResult() As Boolean
```

**C# syntax**

```
public override bool NextResult()
```

**Returns**

True if there are more result sets, false otherwise. For UltraLite.NET, always returns false.

**Exceptions**

- **ULException class**  
  The ULDataReader object is not opened.

**Remarks**

**UL Ext:** UltraLite.NET does not support batches of SQL statements, so the ULDataReader object is always positioned on the first and only result set. NextResult method calls have no effect.

---

**Read method**

Positions the cursor to the next row, or after the last row if the cursor was already on the last row.

**Visual Basic syntax**

```
Public Overrides Function Read() As Boolean
```

**C# syntax**

```
public override bool Read()
```

**Returns**

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

**Exceptions**

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

**Remarks**

This method is identical to the ULDataReader.MoveNext method.

**See also**

- “ULDataReader.MoveNext method [UltraLite.NET]” on page 257
**Depth property**

Returns the depth of nesting for the current row.

**Visual Basic syntax**

```
Public ReadOnly Overrides Property Depth As Integer
```

**C# syntax**

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

**Exceptions**

- **ULException class**  The ULDataReader object is not opened.

**Remarks**

The outermost table has a depth of zero.

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

---

**FieldCount property**

Returns the number of columns in the cursor.

**Visual Basic syntax**

```
Public ReadOnly Overrides Property FieldCount As Integer
```

**C# syntax**

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

**Returns**

The number of columns in the cursor as an integer. Returns 0 if the cursor is closed.

**Remarks**

This method is identical to the ULCursorSchema.ColumnCount method.

**See also**

- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

---

**HasRows property**

Checks whether the ULDataReader object has one or more rows.

**Visual Basic syntax**

```
Public ReadOnly Overrides Property HasRows As Boolean
```
**C# syntax**
```csharp
public override bool HasRows { get; }
```

**Remarks**
True if the result set has at least one row, false if there are no rows.

**IsBOF property**

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

**Visual Basic syntax**
```
Public ReadOnly Property IsBOF As Boolean
```

**C# syntax**
```csharp
public bool IsBOF { get; }
```

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

**Remarks**
True if the current row position is before the first row, false otherwise.

**IsClosed property**

Checks whether the cursor is currently open.

**Visual Basic syntax**
```
Public ReadOnly Overrides Property IsClosed As Boolean
```

**C# syntax**
```csharp
public override bool IsClosed { get; }
```

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

**IsEOF property**

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

**Visual Basic syntax**
```
Public ReadOnly Property IsEOF As Boolean
```
C# syntax
    public bool IsEOF {get;}

Exceptions
    ● ULException class  A SQL error occurred.

Remarks
    True if the current row position is after the last row, false otherwise.

RecordsAffected property
    Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.

Visual Basic syntax
    Public ReadOnly Overrides Property RecordsAffected As Integer

C# syntax
    public override int RecordsAffected {get;}

Remarks
    For SELECT statements or CommandType.TableDirect tables, this value is -1.
    The number of rows changed, inserted, or deleted by execution of the SQL statement.

See also
    ● System.Data.CommandType

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

Visual Basic syntax
    Public ReadOnly Property RowCount As Integer

C# syntax
    public int RowCount {get;}

Exceptions
    ● ULException class  A SQL error occurred.

Remarks
    The number of rows in the cursor.
You can use the RowCount method to decide when to delete old rows to save space. Old rows can be deleted from the UltraLite database without being deleted from the consolidated database using the ULConnection.StopSynchronizationDelete method.

See also

- “ULConnection.StartSynchronizationDelete method [UltraLite.NET]” on page 150
- “ULConnection.StopSynchronizationDelete method [UltraLite.NET]” on page 150

Schema property

UL Ext: Holds the schema of this cursor.

Visual Basic syntax

Public ReadOnly Property Schema As ULCursorSchema

C# syntax

public ULCursorSchema Schema {get;}

Remarks

For result sets, the ULResultSetSchema object representing the schema of the result set. For tables, the ULTableSchema object representing the schema of the table.

This property represents the complete schema of the cursor, including UltraLite.NET extended information which is not represented in the results from the ULDataReader.GetSchemaTable method.

See also

- “ULTableSchema class [UltraLite.NET]” on page 421
- “ULDataReader.GetSchemaTable method [UltraLite.NET]” on page 249
- “ULResultSetSchema class [UltraLite.NET]” on page 362

this property

Returns the value of the specified column in its native format.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>this[int] property</td>
<td>Returns the value of the specified column in its native format.</td>
</tr>
<tr>
<td>this[string] property</td>
<td>Returns the value of the specified named column in its native format.</td>
</tr>
</tbody>
</table>

this[int] property

Returns the value of the specified column in its native format.
Visual Basic syntax

Public ReadOnly Overrides Property Item(ByVal colID As Integer) As Object

C# syntax

public override object this[int colID] {get;}

Parameters

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

Returns

The column value as the .NET type most appropriate for the column or DBNull if column is NULL.

Exceptions

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

Remarks

In C#, this property is the indexer for the ULDataReader class.

This method is identical in functionality to the ULDataReader.GetValue(int) method.

See also

- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.GetValue method [UltraLite.NET]” on page 254
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260

**this[string] property**

Returns the value of the specified named column in its native format.

Visual Basic syntax

Public ReadOnly Overrides Property Item(ByVal name As String) As Object

C# syntax

public override object this[string name] {get;}

Parameters

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

Returns

The column value as the .NET type most appropriate for the column or DBNull if column is NULL.
Exceptions

● **ULException class** A SQL error occurred.

Remarks

In C#, this property is the indexer for the ULDataReader object.

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, the MyTable.ID value 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.GetOrdinal method [UltraLite.NET]” on page 247
● “ULDataReader.GetValue method [UltraLite.NET]” on page 254
● “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243

**ULException class**

Represents a SQL error returned by the UltraLite.NET database.

Visual Basic syntax

```
Public NotInheritable Class ULException
    Inherits System.ApplicationException
```

C# syntax

```
public sealed class ULException : System.ApplicationException
```

Base classes

● System.ApplicationException

Members

All members of the ULException class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetObjectData method</td>
<td>Populates a SerializationInfo object with the data needed to serialize this ULException object.</td>
</tr>
<tr>
<td>NativeError property</td>
<td>Returns the SQLCODE value returned by the database.</td>
</tr>
</tbody>
</table>
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source property</td>
<td>Returns the name of the provider that generated the error.</td>
</tr>
</tbody>
</table>

### Remarks

The SQLCODE denoting the error is returned in the NativeError property.

This class is not serializable under the .NET Compact Framework.

### See also

- “ULException.NativeError property [UltraLite.NET]” on page 266
- “SQL Anywhere error messages sorted by SQLCODE” [Error Messages]

### GetObjectData method

Populates a SerializationInfo object with the data needed to serialize this ULException object.

#### Visual Basic syntax

```vbnet
Public Overrides Sub GetObjectData(
    ByVal info As SerializationInfo,
    ByVal context As StreamingContext
)
```

#### C# syntax

```csharp
public override void GetObjectData(
    SerializationInfo info,
    StreamingContext context
)
```

#### Parameters

- **info**  The SerializationInfo object to populate with data.
- **context**  The destination for this serialization.

#### Remarks

This method is not supported under the .NET Compact Framework.

### NativeError property

Returns the SQLCODE value returned by the database.

#### Visual Basic syntax

```vbnet
Public ReadOnly Property NativeError As ULSQLCode
```

### GetObjectData method

Populates a SerializationInfo object with the data needed to serialize this ULException object.

#### Visual Basic syntax

```vbnet
Public Overrides Sub GetObjectData(
    ByVal info As SerializationInfo,
    ByVal context As StreamingContext
)
```

#### C# syntax

```csharp
public override void GetObjectData(
    SerializationInfo info,
    StreamingContext context
)
```

#### Parameters

- **info**  The SerializationInfo object to populate with data.
- **context**  The destination for this serialization.

#### Remarks

This method is not supported under the .NET Compact Framework.

### NativeError property

Returns the SQLCODE value returned by the database.

#### Visual Basic syntax

```vbnet
Public ReadOnly Property NativeError As ULSQLCode
```
C# syntax

```csharp
public ULSQLCode NativeError {get;}
```

Remarks
The ULSQLCode value returned by the database.

### Source property

Returns the name of the provider that generated the error.

**Visual Basic syntax**

```vbnet
Public ReadOnly Overrides Property Source As String
```

**C# syntax**

```csharp
public override string Source {get;}
```

Remarks
The string value identifying UltraLite.NET as the provider.

## ULFactory class

Represents a set of methods for creating instances of the iAnywhere.Data.UltraLite provider’s implementation of the data source classes.

**Visual Basic syntax**

```vbnet
```

**C# syntax**

```csharp
```

### Base classes


### Members

All members of the ULFactory class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CanCreateDataSourceEnumerator property</td>
<td>Returns false to indicate the UltraLite.NET does not support the DbDataSourceEnumerator class.</td>
</tr>
<tr>
<td>Instance field</td>
<td>Represents the singleton instance of the ULFactory class.</td>
</tr>
</tbody>
</table>

**Remarks**

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

ADO.NET 2.0 adds two new classes, the System.Data.Common.DbProviderFactories class and the System.Data.Common.DbProviderFactory class, to make provider independent code easier to write. To use them with UltraLite.NET specify iAnywhere.Data.UltraLite as the provider invariant name passed to the GetFactory method. For example:

```vbnet
' Visual Basic
Dim factory As DbProviderFactory = DbProviderFactories.GetFactory( "iAnywhere.Data.UltraLite" )
Dim conn As DbConnection = factory.CreateConnection()
```

The following code is the C# language equivalent:

```csharp
// 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 "UltraLite.NET does not support the CreateCommandBuilder, CreateDataSourceEnumerator, and CreatePermission methods."
CreateCommand method


Visual Basic syntax

Public Overrides Function CreateCommand() As DbCommand

C# syntax

public override DbCommand CreateCommand()

Returns

A new ULCommand instance typed as DbCommand.

See also

- “ULCommand class [UltraLite.NET]” on page 71

CreateCommandBuilder method


Visual Basic syntax

Public Overrides Function CreateCommandBuilder() As DbCommandBuilder

C# syntax

public override DbCommandBuilder CreateCommandBuilder()

Returns

A new ULCommandBuilder instance typed as DbCommandBuilder.

See also

- “ULCommandBuilder class [UltraLite.NET]” on page 107

CreateConnection method


Visual Basic syntax

Public Overrides Function CreateConnection() As DbConnection
C# syntax

```csharp
public override DbConnection CreateConnection()
```

**Returns**
A new ULConnection instance typed as DbConnection.

**See also**
- “ULConnection class [UltraLite.NET]” on page 118

## CreateConnectionStringBuilder method

**Visual Basic syntax**

```vbnet
Public Overrides Function CreateConnectionStringBuilder() As DbConnectionStringBuilder
```

**C# syntax**

```csharp
public override DbConnectionStringBuilder CreateConnectionStringBuilder()
```

**Returns**
A new ULConnectionStringBuilder instance typed as DbConnectionStringBuilder.

**See also**
- “ULConnectionStringBuilder class [UltraLite.NET]” on page 172

## CreateDataAdapter method

**Visual Basic syntax**

```vbnet
Public Overrides Function CreateDataAdapter() As DbDataAdapter
```

**C# syntax**

```csharp
public override DbDataAdapter CreateDataAdapter()
```

**Returns**
A new ULDataAdapter instance typed as DbDataAdapter.
See also

- “ULDataAdapter class [UltraLite.NET]” on page 204

**CreateParameter method**


**Visual Basic syntax**

```vbnet
Public Overrides Function CreateParameter() As DbParameter
```

**C# syntax**

```csharp
public override DbParameter CreateParameter()
```

**Returns**

A new ULParameter instance typed as DbParameter.

**See also**

- “ULParameter class [UltraLite.NET]” on page 307

**CanCreateDataSourceEnumerator property**

Returns false to indicate the UltraLite.NET does not support the DbDataSourceEnumerator class.

**Visual Basic syntax**

```vbnet
Public ReadOnly Overrides Property CanCreateDataSourceEnumerator As Boolean
```

**C# syntax**

```csharp
public override bool CanCreateDataSourceEnumerator {get;}
```

**Remarks**

False to indicate that the ULFactory class does not implement the CreateDataSourceEnumerator method.

**Instance field**

Represents the singleton instance of the ULFactory class.

**Visual Basic syntax**

```vbnet
Public Shared ReadOnly Instance As ULFactory
```

**C# syntax**

```csharp
public static readonly ULFactory Instance;
```
Remarks
The ULFactory class is not available in the .NET Compact Framework 2.0.

ULFactory is a singleton class, which means only this instance of this class can exist.

Normally you would not use this field directly. Instead, you get a reference to this instance of the ULFactory class using the System.Data.Common.DbProviderFactories.GetFactory(String) method.

See also
- “ULFactory class [UltraLite.NET]” on page 267

ULFileTransfer class
UL Ext: Transfers a file from a remote database using the MobiLink server.

Visual Basic syntax
Public NotInheritable Class ULFileTransfer

C# syntax
public sealed class ULFileTransfer

Members
All members of the ULFileTransfer class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULFileTransfer constructor</td>
<td>Initializes a ULFileTransfer object.</td>
</tr>
<tr>
<td>DownloadFile method</td>
<td>Download the file specified by the properties of this object.</td>
</tr>
<tr>
<td>UploadFile method</td>
<td>Upload the file specified by the properties of this object.</td>
</tr>
<tr>
<td>AuthenticationParms property</td>
<td>Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).</td>
</tr>
<tr>
<td>AuthStatus property</td>
<td>Returns the authorization status code for the last file transfer attempt.</td>
</tr>
<tr>
<td>AuthValue property</td>
<td>Returns the return value from custom user authentication synchronization scripts.</td>
</tr>
<tr>
<td>FileAuthCode property</td>
<td>Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.</td>
</tr>
<tr>
<td>FileName property</td>
<td>Specifies the name of the file to download.</td>
</tr>
<tr>
<td>LocalFileName property</td>
<td>Specifies the local file name for the downloaded file.</td>
</tr>
</tbody>
</table>
### ULFileTransfer class

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalPath property</td>
<td>Specifies where to download the file.</td>
</tr>
<tr>
<td>Password property</td>
<td>The MobiLink password for the user specified by the UserName value.</td>
</tr>
<tr>
<td>RemoteKey property</td>
<td>The key that uniquely identifies the MobiLink client to the MobiLink server.</td>
</tr>
<tr>
<td>ResumePartialDownload property</td>
<td>Specifies whether to resume or discard a previous partial download.</td>
</tr>
<tr>
<td>Stream property</td>
<td>Specifies the MobiLink synchronization stream to use for the file transfer.</td>
</tr>
<tr>
<td>StreamErrorCode property</td>
<td>Returns the error reported by the stream itself for the last file transfer attempt.</td>
</tr>
<tr>
<td>StreamErrorSystem property</td>
<td>Returns the stream error system-specific code.</td>
</tr>
<tr>
<td>StreamParms property</td>
<td>Specifies the parameters to configure the synchronization stream.</td>
</tr>
<tr>
<td>TransferredFile property</td>
<td>Checks whether the file was actually downloaded during the last file transfer attempt.</td>
</tr>
<tr>
<td>UserName property</td>
<td>The user name that identifies the MobiLink client to the MobiLink server.</td>
</tr>
<tr>
<td>Version property</td>
<td>Specifies which synchronization script to use.</td>
</tr>
</tbody>
</table>

**Remarks**

You do not need a database connection to perform a file transfer, however, if your application uses an UltraLite database with the UltraLite Engine runtime, you must set the ULDatabaseManager.RuntimeType value to the appropriate value before using this API or any other UltraLite.NET API.

To transfer a file you must set the ULFileTransfer.FileName, ULFileTransfer.Stream, ULFileTransfer.UserName, and ULFileTransfer.Version values.

**See also**

- “ULFileTransfer.FileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.Stream property [UltraLite.NET]” on page 284
- “ULFileTransfer.UserName property [UltraLite.NET]” on page 286
- “ULFileTransfer.Version property [UltraLite.NET]” on page 286
ULFileTransfer constructor

Initializes a ULFileTransfer object.

Visual Basic syntax

Public Sub New()

C# syntax

public ULFileTransfer()

Remarks

The connection must be opened before you can perform any operations against the database.

You do not need a database connection to perform a file transfer, however, if your application uses an UltraLite database with the UltraLite Engine runtime, you must set the ULDatabaseManager.RuntimeType value to the appropriate value before using this API or any other UltraLite.NET API.

The ULFileTransfer object needs to have the ULFileTransfer.FileName, ULFileTransfer.Stream, ULFileTransfer.UserName, and ULFileTransfer.Version values set before it can transfer a file.

See also

- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277
- “ULFileTransfer.FileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.Stream property [UltraLite.NET]” on page 284
- “ULFileTransfer.UserName property [UltraLite.NET]” on page 286
- “ULFileTransfer.Version property [UltraLite.NET]” on page 286

DownloadFile method

Download the file specified by the properties of this object.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DownloadFile() method</td>
<td>Download the file specified by the properties of this object.</td>
</tr>
<tr>
<td>DownloadFile(ULFileTransfer-ProgressListener) method</td>
<td>Download the file specified by the properties of this object with progress events posted to the specified listener.</td>
</tr>
</tbody>
</table>

DownloadFile() method

Download the file specified by the properties of this object.
Visual Basic syntax

Public Function DownloadFile() As Boolean

C# syntax

public bool DownloadFile()

Returns

True if successful, false otherwise (check the ULFileTransfer.StreamErrorCode value and other status properties for reason).

Remarks

The file specified by the ULFileTransfer.FileName value is downloaded by the MobiLink server to the ULFileTransfer.LocalPath value using the ULFileTransfer.Stream, ULFileTransfer.UserName, ULFileTransfer.Password, and ULFileTransfer.Version values.

To avoid file corruption, UltraLite.NET downloads to a temporary file and only replaces the local file once the download has completed. Other properties that affect the download are: ULFileTransfer.LocalFileName, ULFileTransfer.AuthenticationParms, and ULFileTransfer.ResumePartialDownload.

A detailed result status is reported in this object's ULFileTransfer.AuthStatus, ULFileTransfer.AuthValue, ULFileTransfer.FileAuthCode, ULFileTransfer.TransferredFile, ULFileTransfer.StreamErrorCode, and ULFileTransfer.StreamErrorSystem values.

See also

- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.FileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.LocalPath property [UltraLite.NET]” on page 282
- “ULFileTransfer.Stream property [UltraLite.NET]” on page 284
- “ULFileTransfer.UserName property [UltraLite.NET]” on page 286
- “ULFileTransfer.Password property [UltraLite.NET]” on page 282
- “ULFileTransfer.Version property [UltraLite.NET]” on page 286
- “ULFileTransfer.LocalFileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.AuthenticationParms property [UltraLite.NET]” on page 279
- “ULFileTransfer.ResumePartialDownload property [UltraLite.NET]” on page 283
- “ULFileTransfer.AuthStatus property [UltraLite.NET]” on page 280
- “ULFileTransfer.AuthValue property [UltraLite.NET]” on page 280
- “ULFileTransfer.FileAuthCode property [UltraLite.NET]” on page 280
- “ULFileTransfer.TransferredFile property [UltraLite.NET]” on page 285
- “ULFileTransfer.StreamErrorCode property [UltraLite.NET]” on page 284
- “ULFileTransfer.StreamErrorSystem property [UltraLite.NET]” on page 285
- “ULFileTransfer.StreamErrorCode property [UltraLite.NET]” on page 284
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277
DownloadFile(ULFileTransferProgressListener) method

Download the file specified by the properties of this object with progress events posted to the specified listener.

Visual Basic syntax

Public Function DownloadFile(  
    ByVal listener As ULFileTransferProgressListener  
) As Boolean

C# syntax

public bool DownloadFile(ULFileTransferProgressListener listener)

Parameters

- **listener** The object that receives file transfer progress events.

Returns

True if successful, false otherwise (check the ULFileTransfer.StreamErrorCode value and other status properties for reason).

Remarks

The file specified by the ULFileTransfer.FileName value is downloaded by the MobiLink server to the ULFileTransfer.LocalPath value using the ULFileTransfer.Stream, ULFileTransfer.UserName, ULFileTransfer.Password, and ULFileTransfer.Version values.

To avoid file corruption, UltraLite.NET downloads to a temporary file and only replaces the local file once the download has completed. Other properties that affect the download are: ULFileTransfer.LocalFileName, ULFileTransfer.AuthenticationParms, and ULFileTransfer.ResumePartialDownload.

A detailed result status is reported in this object's ULFileTransfer.AuthStatus, ULFileTransfer.AuthValue, ULFileTransfer.FileAuthCode, ULFileTransfer.TransferredFile, ULFileTransfer.StreamErrorCode, and ULFileTransfer.StreamErrorSystem values.

Errors may result in no data being sent to the listener.
See also

- “ULFileTransfer.FileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.LocalPath property [UltraLite.NET]” on page 282
- “ULFileTransfer.Stream property [UltraLite.NET]” on page 284
- “ULFileTransfer.UserName property [UltraLite.NET]” on page 286
- “ULFileTransfer.Password property [UltraLite.NET]” on page 282
- “ULFileTransfer.Version property [UltraLite.NET]” on page 286
- “ULFileTransfer.LocalFileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.AuthenticationParms property [UltraLite.NET]” on page 279
- “ULFileTransfer.ResumePartialDownload property [UltraLite.NET]” on page 279
- “ULFileTransfer.AuthStatus property [UltraLite.NET]” on page 280
- “ULFileTransfer.AuthValue property [UltraLite.NET]” on page 280
- “ULFileTransfer.FileAuthCode property [UltraLite.NET]” on page 280
- “ULFileTransfer.TransferredFile property [UltraLite.NET]” on page 285
- “ULFileTransfer.StreamErrorCode property [UltraLite.NET]” on page 284
- “ULFileTransfer.StreamErrorSystem property [UltraLite.NET]” on page 285
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277

UploadFile method

Upload the file specified by the properties of this object.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UploadFile() method</td>
<td>Upload the file specified by the properties of this object.</td>
</tr>
<tr>
<td>UploadFile(ULFileTransfer-ProgressListener) method</td>
<td>Upload the file specified by the properties of this object with progress events posted to the specified listener.</td>
</tr>
</tbody>
</table>

UploadFile() method

Upload the file specified by the properties of this object.

Visual Basic syntax

```vbnet
Public Function UploadFile() As Boolean
```

C# syntax

```csharp
public bool UploadFile()
```

Returns

True if successful, false otherwise (check the ULFileTransfer.StreamErrorCode value and other status properties for reason).
Remarks

The file specified by the ULFileTransfer.FileName value is uploaded to the MobiLink server from the ULFileTransfer.LocalPath value using the ULFileTransfer.Stream, ULFileTransfer.UserName, ULFileTransfer.Password, and ULFileTransfer.Version values.

A detailed result status is reported in this object's ULFileTransfer.AuthStatus, ULFileTransfer.AuthValue, ULFileTransfer.FileAuthCode, ULFileTransfer.TransferredFile, ULFileTransfer.StreamErrorCode, and ULFileTransfer.StreamErrorSystem values.

See also

- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277
- “ULFileTransfer.FileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.LocalPath property [UltraLite.NET]” on page 282
- “ULFileTransfer.Stream property [UltraLite.NET]” on page 284
- “ULFileTransfer.UserName property [UltraLite.NET]” on page 286
- “ULFileTransfer.Password property [UltraLite.NET]” on page 282
- “ULFileTransfer.Version property [UltraLite.NET]” on page 286
- “ULFileTransfer.LocalFileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.AuthenticationParms property [UltraLite.NET]” on page 279
- “ULFileTransfer.ResumePartialDownload property [UltraLite.NET]” on page 283
- “ULFileTransfer.AuthStatus property [UltraLite.NET]” on page 280
- “ULFileTransfer.AuthValue property [UltraLite.NET]” on page 280
- “ULFileTransfer.FileAuthCode property [UltraLite.NET]” on page 280
- “ULFileTransfer.TransferredFile property [UltraLite.NET]” on page 285
- “ULFileTransfer.StreamErrorCode property [UltraLite.NET]” on page 284
- “ULFileTransfer.StreamErrorSystem property [UltraLite.NET]” on page 285
- “ULFileTransfer.StreamErrorCode property [UltraLite.NET]” on page 284

UploadFile(ULFileTransferProgressListener) method

Upload the file specified by the properties of this object with progress events posted to the specified listener.

Visual Basic syntax

Public Function UploadFile(
    ByVal listener As ULFileTransferProgressListener
) As Boolean

C# syntax

public bool UploadFile(ULFileTransferProgressListener listener)

Parameters

- listener The object that receives file transfer progress events.
Returns

True if successful, false otherwise (check the ULFileTransfer.StreamErrorCode value and other status properties for reason).

Remarks

The file specified by the ULFileTransfer.FileName value is uploaded to the MobiLink server from the ULFileTransfer.LocalPath value using the ULFileTransfer.Stream, ULFileTransfer.UserName, ULFileTransfer.Password, and ULFileTransfer.Version values.

A detailed result status is reported in this object's ULFileTransfer.AuthStatus, ULFileTransfer.AuthValue, ULFileTransfer.FileAuthCode, ULFileTransfer.TransferredFile, ULFileTransfer.StreamErrorCode, and ULFileTransfer.StreamErrorSystem values.

Errors may result in no data being sent to the listener.

See also

- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277
- “ULFileTransfer.FileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.LocalPath property [UltraLite.NET]” on page 282
- “ULFileTransfer.Stream property [UltraLite.NET]” on page 284
- “ULFileTransfer.UserName property [UltraLite.NET]” on page 286
- “ULFileTransfer.Password property [UltraLite.NET]” on page 282
- “ULFileTransfer.Version property [UltraLite.NET]” on page 286
- “ULFileTransfer.LocalFileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.AuthenticationParms property [UltraLite.NET]” on page 279
- “ULFileTransfer.ResumePartialDownload property [UltraLite.NET]” on page 279
- “ULFileTransfer.AuthStatus property [UltraLite.NET]” on page 280
- “ULFileTransfer.AuthValue property [UltraLite.NET]” on page 280
- “ULFileTransfer.FileAuthCode property [UltraLite.NET]” on page 280
- “ULFileTransfer.TransferredFile property [UltraLite.NET]” on page 285
- “ULFileTransfer.StreamErrorCode property [UltraLite.NET]” on page 284
- “ULFileTransfer.StreamErrorSystem property [UltraLite.NET]” on page 285
- “ULFileTransfer.StreamErrorCode property [UltraLite.NET]” on page 284

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Visual Basic syntax

    Public Property AuthenticationParms As String()

C# syntax

    public String[] AuthenticationParms {get;set;}

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 279
Remarks

An array of strings, each containing an authentication parameter (null array entries result in a synchronization error). The default is a null reference (Nothing in Visual Basic), meaning no authentication parameters.

Only the first 255 strings are used and each string should be no longer than the MobiLink server's limit for authentication parameters (currently 4000 UTF8 bytes).

AuthStatus property

Returns the authorization status code for the last file transfer attempt.

Visual Basic syntax

Public ReadOnly Property AuthStatus As ULAuthStatusCode

C# syntax

public ULAuthStatusCode AuthStatus {get;}

Remarks

One of the ULAuthStatusCode values denoting the authorization status for the last file transfer attempt.

See also

● “ULAuthStatusCode enumeration [UltraLite.NET]” on page 440

AuthValue property

Returns the return value from custom user authentication synchronization scripts.

Visual Basic syntax

Public ReadOnly Property AuthValue As Long

C# syntax

public long AuthValue {get;}

Remarks

A long integer returned from custom user authentication synchronization scripts.

FileAuthCode property

Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.

Visual Basic syntax

Public ReadOnly Property FileAuthCode As UShort
C# syntax

```csharp
public ushort FileAuthCode {get;}
```

Remarks
An unsigned short integer returned from the authenticate_file_transfer script for the last file transfer attempt.

**FileName property**

Specifies the name of the file to download.

Visual Basic syntax

```vbnet
Public Property FileName As String
```

C# syntax

```csharp
public string FileName {get;set;}
```

Remarks
A string specifying the name of the file as recognized by the MobiLink server. This property has no default value, and must be explicitly set.

The FileName value is the name of the file on the server running MobiLink. MobiLink first searches for this file in the UserName subdirectory and then in the root directory (the root directory is specified via the MobiLink server's -ftr option). The FileName value must not include any drive or path information so that the MobiLink server can find it. For example, "myfile.txt" is valid, but "somedir\myfile.txt", "..\myfile.txt", and "c:\myfile.txt" are all invalid.

See also
- “ULFileTransfer.LocalFileName property [UltraLite.NET]” on page 281
- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277

**LocalFileName property**

Specifies the local file name for the downloaded file.

Visual Basic syntax

```vbnet
Public Property LocalFileName As String
```

C# syntax

```csharp
public string LocalFileName {get;set;}
```

Remarks
A string specifying the local file name for the downloaded file. The FileName value is used if the value is a null reference (Nothing in Visual Basic). The default is a null reference (Nothing in Visual Basic).
See also

- “ULFileTransfer.FileName property [UltraLite.NET]” on page 281

LocalPath property

Specifies where to download the file.

Visual Basic syntax

Public Property LocalPath As String

C# syntax

public string LocalPath {get;set;}

Remarks

A string specifying the local directory of the file. The default is a null reference (Nothing in Visual Basic).

The default local directory varies depending on the device's operating system:

- For Windows Mobile devices, if the LocalPath value is a null reference (Nothing in Visual Basic), the file is stored in the root (\) directory.
- For desktop applications, if the LocalPath value is a null reference (Nothing in Visual Basic), the file is stored in the current directory.

See also

- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277

Password property

The MobiLink password for the user specified by the UserName value.

Visual Basic syntax

Public Property Password As String

C# syntax

public string Password {get;set;}

Remarks

A string specifying the MobiLink password. The default is a null reference (Nothing in Visual Basic), meaning no password is specified.

The MobiLink user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink server.
See also

- “ULFileTransfer.UserName property [UltraLite.NET]” on page 286
- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277

RemoteKey property

The key that uniquely identifies the MobiLink client to the MobiLink server.

Visual Basic syntax

Public Property RemoteKey As String

C# syntax

public string RemoteKey {get;set;}

Remarks

A string specifying the remote key. This property has no default value, and must be explicitly set.

The MobiLink server passes this value to various scripts to uniquely identify this client.

See also

- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Visual Basic syntax

Public Property ResumePartialDownload As Boolean

C# syntax

public bool ResumePartialDownload {get;set;}

Remarks

True to resume a previous partial download, false to discard a previous partial download. The default is false.

UltraLite.NET can use the ULFileTransferListener object to restart downloads that fail because of communication errors or user aborts. UltraLite.NET processes the download as it is received. If a download is interrupted, then the partially download file is retained and can be resumed during the next file transfer.

If the file has been updated on the server, a partial download is discarded and a new download starts.
See also

- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277

**Stream property**

Specifies the MobiLink synchronization stream to use for the file transfer.

**Visual Basic syntax**

```vbnet
Public Property Stream As ULSStreamType
```

**C# syntax**

```csharp
public ULSStreamType Stream {get;set;}
```

**Remarks**

One of the ULSStreamType values specifying the type of synchronization stream to use. The default is the ULSStreamType.TCPIP value.

Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. These parameters are supplied by the ULFileTransfer.StreamParms value.

If the stream type is set to a value that is invalid for the platform, the stream type is set to the ULSStreamType.TCPIP value.

See also

- “ULStreamType enumeration [UltraLite.NET]” on page 448
- “ULFileTransfer.StreamParms property [UltraLite.NET]” on page 285
- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277

**StreamErrorCode property**

Returns the error reported by the stream itself for the last file transfer attempt.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property StreamErrorCode As ULSStreamErrorCode
```

**C# syntax**

```csharp
public ULSStreamErrorCode StreamErrorCode {get;}
```

**Remarks**

One of the ULSStreamErrorCode values denoting the error reported by the stream itself. The ULSStreamErrorCode.NONE value if no error occurred.
**StreamErrorSystem property**

Returns the stream error system-specific code.

Visual Basic syntax

```vbnet
Public ReadOnly Property StreamErrorSystem As Integer
```

C# syntax

```csharp
public int StreamErrorSystem {get;}
```

**Remarks**

An integer denoting the stream error system-specific code.

**StreamParms property**

Specifies the parameters to configure the synchronization stream.

Visual Basic syntax

```vbnet
Public Property StreamParms As String
```

C# syntax

```csharp
public string StreamParms {get;set;}
```

**Remarks**

A string, in the form of a semicolon-separated list of keyword-value pairs, specifying the parameters for the stream. The default is a null reference. (Nothing in Visual Basic)

The StreamParms value 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.Stream property [UltraLite.NET]” on page 284
- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277
- “ULStreamType enumeration [UltraLite.NET]” on page 448
- “UltraLite network protocol options” [UltraLite - Database Management and Reference]

**TransferredFile property**

Checks whether the file was actually downloaded during the last file transfer attempt.

Visual Basic syntax

```vbnet
Public ReadOnly Property TransferredFile As Boolean
```
C# syntax

```csharp
public bool TransferredFile {get;}
```

Remarks
True if the file was downloaded, false otherwise.

If the file is already up-to-date when the DownloadFile or UploadFile method is invoked, it returns true, but TransferredFile is false. If an error occurs and the DownloadFile or UploadFile method returns false, then TransferredFile is false.

See also
- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277

**UserName property**

The user name that identifies the MobiLink client to the MobiLink server.

Visual Basic syntax

```vbnet
Public Property UserName As String
```

C# syntax

```csharp
public string UserName {get;set;}
```

Remarks
A string specifying the user name. This property has no default value, and must be explicitly set.

The MobiLink server uses this value to locate the file to download. The MobiLink user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink server.

See also
- “ULFileTransfer.Password property [UltraLite.NET]” on page 282
- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277

**Version property**

Specifies which synchronization script to use.

Visual Basic syntax

```vbnet
Public Property Version As String
```

C# syntax

```csharp
public string Version {get;set;}
```
Remarks
A string specifying the version of the synchronization script to use. This property has no default value, and must be explicitly set.

Each synchronization script in the consolidated database is marked with a version string. The version string allows an UltraLite application to choose from a set of synchronization scripts.

See also
- “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274
- “ULFileTransfer.UploadFile method [UltraLite.NET]” on page 277

ULFileTransferProgressData class

UL Ext: Returns file transfer progress monitoring data.

Visual Basic syntax
Public Class ULFileTransferProgressData

C# syntax
public class ULFileTransferProgressData

Members
All members of the ULFileTransferProgressData class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BytesReceived property</td>
<td>Returns the number of bytes received so far.</td>
</tr>
<tr>
<td>FileSize property</td>
<td>Returns the size of the file being transferred.</td>
</tr>
<tr>
<td>Flags property</td>
<td>Returns the current file transfer flags indicating additional information relating to the current state.</td>
</tr>
<tr>
<td>ResumedAtSize property</td>
<td>Returns the point in the file where the transfer was resumed.</td>
</tr>
<tr>
<td>FLAG_IS_BLOCKING field</td>
<td>A flag indicating that the file transfer is blocked awaiting a response from the MobiLink server.</td>
</tr>
</tbody>
</table>

See also
- “ULFileTransferProgressListener interface [UltraLite.NET]” on page 289

BytesReceived property
Returns the number of bytes received so far.
Visual Basic syntax

Public ReadOnly Property BytesReceived As ULong

C# syntax

public ulong BytesReceived {get;}

Remarks
The number of bytes received so far.

FileSize property

Returns the size of the file being transferred.

Visual Basic syntax

Public ReadOnly Property FileSize As ULong

C# syntax

public ulong FileSize {get;}

Remarks
The size of the file in bytes.

Flags property

Returns the current file transfer flags indicating additional information relating to the current state.

Visual Basic syntax

Public ReadOnly Property Flags As Integer

C# syntax

public int Flags {get;}

Remarks
An integer containing a combination of flags or'ed together.

See also

● “ULFileTransferProgressData.FLAG_IS_BLOCKING field [UltraLite.NET]” on page 289

ResumedAtSize property

Returns the point in the file where the transfer was resumed.
Visual Basic syntax
  Public ReadOnly Property ResumedAtSize As ULong

C# syntax
  public ulong ResumedAtSize {get;}

Remarks
  The number of bytes transferred previously.

**FLAG_IS_BLOCKING field**
  A flag indicating that the file transfer is blocked awaiting a response from the MobiLink server.

Visual Basic syntax
  Public Const FLAG_IS_BLOCKING As Integer

C# syntax
  public const int FLAG_IS_BLOCKING;

ULFileTransferProgressListener interface
  **UL Ext:** The listener interface for receiving file transfer progress events.

Visual Basic syntax
  Public Interface ULFileTransferProgressListener

C# syntax
  public interface ULFileTransferProgressListener

Members
  All members of the ULFileTransferProgressListener interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileTransferProgressed method</td>
<td>Invoked during a file transfer to inform the user of progress.</td>
</tr>
</tbody>
</table>

See also
  ● “ULFileTransfer.DownloadFile method [UltraLite.NET]” on page 274

**FileTransferProgressed method**
  Invoked during a file transfer to inform the user of progress.
Visual Basic syntax

Public Function FileTransferProgressed(
    ByVal data As ULFileTransferProgressData
) As Boolean

C# syntax

public bool FileTransferProgressed(ULFileTransferProgressData data)

Parameters

● data  A ULFileTransferProgressData object containing the latest file transfer progress data.

Returns

This method should return true to cancel the transfer or return false to continue.

Remarks

This method should return true to cancel the transfer or return false to continue.

No UltraLite.NET API methods should be invoked during a FileTransferProgressed call.

See also

● “ULFileTransferProgressData class [UltraLite.NET]” on page 287

ULIndexSchema class

UL Ext: Represents the schema of an UltraLite table index.

Visual Basic syntax

Public NotInheritable Class ULIndexSchema

C# syntax

public sealed class ULIndexSchema

Members

All members of the ULIndexSchema class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetColumnName method</td>
<td>Returns the name of the colOrdinalInIndex 'th column in this index.</td>
</tr>
<tr>
<td>IsColumnDescending method</td>
<td>Checks whether the named column is used in descending order by the index.</td>
</tr>
<tr>
<td>ColumnCount property</td>
<td>Returns the number of columns in the index.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsForeignKey property</td>
<td>Checks whether the index is a foreign key.</td>
</tr>
<tr>
<td>IsForeignKeyCheckOnCommit property</td>
<td>Checks whether referential integrity for the foreign key is performed on commits or on inserts and updates.</td>
</tr>
<tr>
<td>IsForeignKeyNullable property</td>
<td>Checks whether the foreign key is nullable.</td>
</tr>
<tr>
<td>IsOpen property</td>
<td>Determines whether the index schema is open or closed.</td>
</tr>
<tr>
<td>IsPrimaryKey property</td>
<td>Checks whether the index is the primary key.</td>
</tr>
<tr>
<td>IsUniqueIndex property</td>
<td>Checks whether the index is unique.</td>
</tr>
<tr>
<td>IsUniqueKey property</td>
<td>Checks whether the index is a unique key.</td>
</tr>
<tr>
<td>Name property</td>
<td>Returns the name of the index.</td>
</tr>
<tr>
<td>ReferencedIndexName property</td>
<td>The name of the referenced primary index if the index is a foreign key.</td>
</tr>
<tr>
<td>ReferencedTableName property</td>
<td>The name of the referenced primary table if the index is a foreign key.</td>
</tr>
</tbody>
</table>

**Remarks**

There is no constructor for this class. Index schemas are created using the ULTableSchema.PrimaryKey, ULTableSchema.GetIndex(string), and ULTableSchema.GetOptimalIndex(int) methods.

**See also**
- “ULTableSchema.PrimaryKey property [UltraLite.NET]” on page 433
- “ULTableSchema.GetIndex method [UltraLite.NET]” on page 425
- “ULTableSchema.GetOptimalIndex method [UltraLite.NET]” on page 426
- “ULTableSchema class [UltraLite.NET]” on page 421

**GetColumnName method**

Returns the name of the \textit{colOrdinalInIndex} 'th column in this index.

**Visual Basic syntax**

```vbnet
Public Function GetColumnName(
    ByVal colOrdinalInIndex As Short
) As String
```

**C# syntax**

```csharp
public string GetColumnName(short colOrdinalInIndex)
```
Parameters
● **colOrdinalInIndex**  The ordinal of the desired column in the index. The value must be in the range [1,ColumnCount].

Returns
The name of the column.

Exceptions
● **ULException class**  A SQL error occurred.

Remarks
Column ordinals and count may change during a schema upgrade. Column ordinals from an index are different than the column IDs in a table or another index, even if they refer to the same physical column in a particular table.

See also
● “ULIndexSchema.ColumnName property [UltraLite.NET]” on page 291

**IsColumnDescending method**
Checks whether the named column is used in descending order by the index.

Visual Basic syntax
    Public Function IsColumnDescending(ByVal name As String) As Boolean

C# syntax
    public bool IsColumnDescending(string name)

Parameters
● **name**  The name of the column.

Returns
True if the column is used in descending order, false if the column is used in ascending order.

Exceptions
● **ULException class**  A SQL error occurred.

See also
● “ULIndexSchema.GetColumnName method [UltraLite.NET]” on page 291
● “ULIndexSchema.ColumnName property [UltraLite.NET]” on page 293
ColumnCount property

Returns the number of columns in the index.

Visual Basic syntax
Public ReadOnly Property ColumnCount As Short

C# syntax
public short ColumnCount {get;}

Remarks
The number of columns in the index.

Column ordinals in indexes range from 1 to the ColumnCount value, inclusively.

Column ordinals and count may change during a schema upgrade. Column ordinals from an index are different than the column IDs in a table or another index, even if they refer to the same physical column in a particular table.

IsForeignKey property

Checks whether the index is a foreign key.

Visual Basic syntax
Public ReadOnly Property IsForeignKey As Boolean

C# syntax
public bool IsForeignKey {get;}

Exceptions
● ULEException class A SQL error occurred.

Remarks
True if the index is the foreign key, false if the index is not the foreign key.

Columns in a foreign key may reference another table's non-null, unique index.

IsForeignKeyCheckOnCommit property

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

Visual Basic syntax
Public ReadOnly Property IsForeignKeyCheckOnCommit As Boolean
C# syntax
   public bool IsForeignKeyCheckOnCommit {get;}

Exceptions
   ●  ULException class  A SQL error occurred (including index is not a foreign key).

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

See also
   ●  “ULIndexSchema.IsForeignKey property [UltraLite.NET]” on page 293

IsForeignKeyNullable property
   Checks whether the foreign key is nullable.

Visual Basic syntax
   Public ReadOnly Property IsForeignKeyNullable As Boolean

C# syntax
   public bool IsForeignKeyNullable {get;}

Exceptions
   ●  ULException class  A SQL error occurred (including index is not a foreign key).

Remarks
   True if the foreign key is nullable, false if the foreign key is not nullable.

See also
   ●  “ULIndexSchema.IsForeignKey property [UltraLite.NET]” on page 293

IsOpen property
   Determines whether the index schema is open or closed.

Visual Basic syntax
   Public ReadOnly Property IsOpen As Boolean

C# syntax
   public bool IsOpen {get;}

Remarks
   True if the index schema is open, otherwise false.
IsPrimaryKey property
Checks whether the index is the primary key.

Visual Basic syntax
Public ReadOnly Property IsPrimaryKey As Boolean

C# syntax
public bool IsPrimaryKey {get;}

Exceptions
● ULException class A SQL error occurred.

Remarks
True if the index is the primary key, false if the index is not the primary key.
Columns in the primary key may not be null.

IsUniqueIndex property
Checks whether the index is unique.

Visual Basic syntax
Public ReadOnly Property IsUniqueIndex As Boolean

C# syntax
public bool IsUniqueIndex {get;}

Exceptions
● ULException class A SQL error occurred.

Remarks
True if the index is unique, false if the index is not unique.
Columns in a unique index may be null.

IsUniqueKey property
Checks whether the index is a unique key.

Visual Basic syntax
Public ReadOnly Property IsUniqueKey As Boolean

C# syntax
public bool IsUniqueKey {get;}

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 295
Exceptions
  ● **ULException class**  A SQL error occurred.

Remarks
  True if the index is a unique key, false if the index is not a unique key.

  Columns in a unique key may not be null.

**Name property**

Returns the name of the index.

**Visual Basic syntax**

```
Public ReadOnly Property Name As String
```

**C# syntax**

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

Exceptions
  ● **ULException class**  A SQL error occurred.

Remarks
  A string specifying the name of the index.

**ReferencedIndexName property**

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

**Visual Basic syntax**

```
Public ReadOnly Property ReferencedIndexName As String
```

**C# syntax**

```
public string ReferencedIndexName {get;}
```

Exceptions
  ● **ULException class**  A SQL error occurred (including index is not a foreign key).

Remarks
  A string specifying the name of the referenced primary index.

See also
  ● “ULIndexSchema.IsForeignKey property [UltraLite.NET]” on page 293
**ReferencedTableName property**

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

**Visual Basic syntax**

`Public ReadOnly Property ReferencedTableName As String`

**C# syntax**

`public string ReferencedTableName {get;}`

**Exceptions**

- **ULException class**  A SQL error occurred (including index is not a foreign key).

**Remarks**

A string specifying the name of the referenced primary table.

**See also**

- “ULIndexSchema.IsForeignKey property [UltraLite.NET]” on page 293

**ULInfoMessageEventArgs class**

Provides data for the ULConnection.InfoMessage event.

**Visual Basic syntax**

`Public NotInheritable Class ULInfoMessageEventArgs Inherits System.EventArgs`

**C# syntax**

`public sealed class ULInfoMessageEventArgs : System.EventArgs`

**Base classes**

- **System.EventArgs**

**Members**

All members of the ULInfoMessageEventArgs class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToString method</td>
<td>The string representation of the ULConnection.InfoMessage event.</td>
</tr>
<tr>
<td>Message property</td>
<td>The informational or warning message string returned by the database.</td>
</tr>
<tr>
<td>NativeError property</td>
<td>The SQLCODE corresponding to the informational message or warning returned by the database.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Source property</td>
<td>The name of the ADO.NET data provider returning the message.</td>
</tr>
<tr>
<td>Empty field (Inherited from System.EventArgs)</td>
<td>Represents an event with no event data.</td>
</tr>
</tbody>
</table>

See also
- “ULConnection.InfoMessage event [UltraLite.NET]” on page 161

**ToString method**
The string representation of the ULConnection.InfoMessage event.

**Visual Basic syntax**
```vbnet
Public Overrides Function ToString() As String
```

**C# syntax**
```csharp
public override string ToString()
```

**Returns**
The informational or warning message string.

**Remarks**
A string representation of the ULConnection.InfoMessage event.

See also
- “ULConnection.InfoMessage event [UltraLite.NET]” on page 161

**Message property**
The informational or warning message string returned by the database.

**Visual Basic syntax**
```vbnet
Public ReadOnly Property Message As String
```

**C# syntax**
```csharp
public string Message {get;}
```

**Remarks**
A string containing the informational or warning message.
**NativeError property**

The SQLCODE corresponding to the informational message or warning returned by the database.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property NativeError As ULSQLCode
```

**C# syntax**

```csharp
public ULSQLCode NativeError {get;}
```

**Remarks**

An informational or warning ULSQLCode value.

**Source property**

The name of the ADO.NET data provider returning the message.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property Source As String
```

**C# syntax**

```csharp
public string Source {get;}
```

**Remarks**

The string "UltraLite.NET Data Provider".

**ULMetaDataCollectionNames class**

Provides a list of constants for use with the ULConnection.GetSchema(String,String[]) method to retrieve metadata collections.

**Visual Basic syntax**

```vbnet
Public NotInheritable Class ULMetaDataCollectionNames
```

**C# syntax**

```csharp
public sealed class ULMetaDataCollectionNames
```

**Members**

All members of the ULMetaDataCollectionNames class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Columns property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the Columns collection.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DataSourceInformation property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the DataSourceInformation collection.</td>
</tr>
<tr>
<td><strong>DataTypes property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the DataTypes collection.</td>
</tr>
<tr>
<td><strong>ForeignKeys property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the ForeignKeys collection.</td>
</tr>
<tr>
<td><strong>IndexColumns property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the IndexColumns collection.</td>
</tr>
<tr>
<td><strong>Indexes property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the Indexes collection.</td>
</tr>
<tr>
<td><strong>MetaDataCollections property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the MetaDataCollections collection.</td>
</tr>
<tr>
<td><strong>Publications property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the Publications collection.</td>
</tr>
<tr>
<td><strong>ReservedWords property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the ReservedWords collection.</td>
</tr>
<tr>
<td><strong>Restrictions property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the Restrictions collection.</td>
</tr>
<tr>
<td><strong>Tables property</strong></td>
<td>Provides a constant for use with the ULConnection.GetSchema(String) method that represents the Tables collection.</td>
</tr>
</tbody>
</table>

**Columns property**

Provides a constant for use with the ULConnection.GetSchema(String) method that represents the Columns collection.

**Visual Basic syntax**

```vbnet
Public Shared ReadOnly Property Columns As String
```

**C# syntax**

```csharp
public string Columns {get;}
```

**Remarks**

A string representing the name of the Columns collection.
See also
● “ULConnection.GetSchema method [UltraLite.NET]” on page 142

Example
The following code fills a DataTable object with the Columns collection.

' Visual Basic
Dim schema As DataTable =_
    conn.GetSchema( ULMetaDataCollectionNames.Columns )

The following code is the C# language equivalent:

// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.Columns );

**DataSourceInformation property**
Provides a constant for use with the ULConnection.GetSchema(String) method that represents the DataSourceInformation collection.

Visual Basic syntax
Public Shared ReadOnly Property DataSourceInformation As String

C# syntax
public string DataSourceInformation {get;}

Remarks
A string representing the name of the DataSourceInformation collection.

See also
● “ULConnection.GetSchema method [UltraLite.NET]” on page 142

Example
The following code fills a DataTable object with the DataSourceInformation collection.

' Visual Basic
Dim schema As DataTable =_
    conn.GetSchema( ULMetaDataCollectionNames.DataSourceInformation )

The following code is the C# language equivalent:

// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.DataSourceInformation );

**DataTypes property**
Provides a constant for use with the ULConnection.GetSchema(String) method that represents the DataTypes collection.
Visual Basic syntax

Public Shared ReadOnly Property DataTypes As String

C# syntax

public string DataTypes {get;}

Remarks

A string representing the name of the DataTypes collection.

See also

● “ULConnection.GetSchema method [UltraLite.NET]” on page 142

Example

The following code fills a DataTable object with the DataTypes collection.

' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.DataTypes )

The following code is the C# language equivalent:

// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.DataTypes );

ForeignKeys property

Provides a constant for use with the ULConnection.GetSchema(String) method that represents the ForeignKeys collection.

Visual Basic syntax

Public Shared ReadOnly Property ForeignKeys As String

C# syntax

public string ForeignKeys {get;}

Remarks

A string representing the name of the ForeignKeys collection.

See also

● “ULConnection.GetSchema method [UltraLite.NET]” on page 142

Example

The following code fills a DataTable object with the ForeignKeys collection.

' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.ForeignKeys )
The following code is the C# language equivalent:

```csharp
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.ForeignKeys );
```

### IndexColumns property

Provides a constant for use with the ULConnection.GetSchema(String) method that represents the IndexColumns collection.

**Visual Basic syntax**

```
Public Shared ReadOnly Property IndexColumns As String
```

**C# syntax**

```
public string IndexColumns {get;}
```

**Remarks**

A string representing the name of the IndexColumns collection.

**See also**

- “ULConnection.GetSchema method [UltraLite.NET]” on page 142

**Example**

The following code fills a DataTable object with the IndexColumns collection.

```vbnet
' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.IndexColumns )
```

The following code is the C# language equivalent:

```csharp
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.IndexColumns );
```

### Indexes property

Provides a constant for use with the ULConnection.GetSchema(String) method that represents the Indexes collection.

**Visual Basic syntax**

```
Public Shared ReadOnly Property Indexes As String
```

**C# syntax**

```
public string Indexes {get;}
```
Remarks
A string representing the name of the Indexes collection.

See also
- “ULConnection.GetSchema method [UltraLite.NET]” on page 142

Example
The following code fills a DataTable object with the Indexes collection.

' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.Indexes )

The following code is the C# language equivalent:

  // C#
  DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.Indexes );

**MetaDataCollections property**
Provides a constant for use with the ULConnection.GetSchema(String) method that represents the MetaDataCollections collection.

Visual Basic syntax
Public Shared ReadOnly Property MetaDataCollections As String

C# syntax
public string MetaDataCollections {get;}

Remarks
A string representing the name of the MetaDataCollections collection.

See also
- “ULConnection.GetSchema method [UltraLite.NET]” on page 142

Example
The following code fills a DataTable object with the MetaDataCollections collection.

' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.MetaDataCollections )

The following code is the C# language equivalent:

    // C#
    DataTable schema =
        conn.GetSchema( ULMetaDataCollectionNames.MetaDataCollections );
Publications property

Provides a constant for use with the ULConnection.GetSchema(String) method that represents the Publications collection.

Visual Basic syntax

Public Shared ReadOnly Property Publications As String

C# syntax

public string Publications {get;}

Remarks

A string representing the name of the Publications collection.

See also

- “ULConnection.GetSchema method [UltraLite.NET]” on page 142

Example

The following code fills a DataTable object with the Publications collection.

' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.Publications )

The following code is the C# language equivalent:

// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.Publications );

ReservedWords property

Provides a constant for use with the ULConnection.GetSchema(String) method that represents the ReservedWords collection.

Visual Basic syntax

Public Shared ReadOnly Property ReservedWords As String

C# syntax

public string ReservedWords {get;}

Remarks

A string representing the name of the ReservedWords collection.

See also

- “ULConnection.GetSchema method [UltraLite.NET]” on page 142
Restrictions property

Provides a constant for use with the ULConnection.GetSchema(String) method that represents the Restrictions collection.

Visual Basic syntax

Public Shared ReadOnly Property Restrictions As String

C# syntax

public string Restrictions {get;}

Remarks

A string representing the name of the Restrictions collection.

See also

● “ULConnection.GetSchema method [UltraLite.NET]” on page 142

Example

The following code fills a DataTable object with the Restrictions collection.

' Visual Basic
Dim schema As DataTable = _
    conn.GetSchema( ULMetaDataCollectionNames.Restrictions )

The following code is the C# language equivalent:

// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.Restrictions );

Tables property

Provides a constant for use with the ULConnection.GetSchema(String) method that represents the Tables collection.

Visual Basic syntax

Public Shared ReadOnly Property Tables As String
C# syntax

```csharp
public string Tables {get;}
```

Remarks

A string representing the name of the Tables collection.

See also

- “ULConnection.GetSchema method [UltraLite.NET]” on page 142

Example

The following code fills a DataTable object with the Tables collection.

```vbnet
' Visual Basic
Dim schema As DataTable =
    conn.GetSchema( ULMetaDataCollectionNames.Tables )
```

```csharp
// C#
DataTable schema =
    conn.GetSchema( ULMetaDataCollectionNames.Tables );
```

ULParameter class

Represents a parameter to a ULCommand object.

Visual Basic syntax

```vbnet
Public NotInheritable Class ULParameter
    Implements System.ICloneable
```

C# syntax

```csharp
public sealed class ULParameter :
    System.Data.Common.DbParameter,
    System.ICloneable
```

Base classes

- System.ICloneable

Members

All members of the ULParameter class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULParameter constructor</td>
<td>Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
ResetDbType method | This method is not supported in UltraLite.NET.
ToString method | Returns the string representation of this instance.
DbType property | Specifies the System.Data.DbType for the parameter.
Direction property | A value indicating whether the parameter is input-only, output-only, bi-directional, or a stored procedure return value parameter.
IsNullable property | Specifies whether the parameter accepts null values.
Offset property | Specifies the offset to the ULParameter.Value.
ParameterName property | Specifies the name of the parameter.
Precision property | Specifies the maximum number of digits used to represent the ULParameter.Value property.
Scale property | Specifies the number of decimal places to which ULParameter.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 object and used for loading or returning the value.
SourceColumnNullMapping property | Specifies whether the source column is nullable.
ULDbType property | Specifies the iAnywhere.Data.UltraLite.ULDbType for the parameter.
Value property | Specifies the value of the parameter.

### Remarks
A ULParameter object can be created directly using one of its many constructors, or using the ULCommand.CreateParameter method. Because of the special treatment of the 0 and 0.0 constants and the way overloaded methods are resolved, it is highly recommended that you explicitly cast constant values to the object type when using the ULParameter(string,object) constructor. For example:

```visualbasic
Dim p As ULParameter = New ULParameter( "", CType( 0, Object ) )
```

The following code is the C# language equivalent:
// C#
ULParameter p = new ULParameter( "", (object)0 );

Parameters (including those created by the ULCommand.CreateCommand parameter method) must be added to a
ULCommand.Parameters collection to be used. All parameters are treated as positional parameters and
are used by a command in the order that they were added.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored.
Only the ULParameter.Value property is important.

See also
● “ULCommand class [UltraLite.NET]” on page 71
● “ULCommand.CreateCommand parameter method [UltraLite.NET]” on page 84
● “ULParameter.ULParameter parameter constructor [UltraLite.NET]” on page 309
● “ULCommand.Parameters property [UltraLite.NET]” on page 105
● “ULParameter.Value property [UltraLite.NET]” on page 321
● System.Data.IDbDataParameter
● System.Data.IDataParameter

ULParameter constructor

Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULParameter() constructor</td>
<td>Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.</td>
</tr>
<tr>
<td>ULParameter(string, object) constructor</td>
<td>Initializes a ULParameter object with the specified parameter name and value.</td>
</tr>
<tr>
<td>ULParameter(string, ULDbType) constructor</td>
<td>Initializes a ULParameter object with the specified parameter name and data type.</td>
</tr>
<tr>
<td>ULParameter(string, ULDbType, int) constructor</td>
<td>Initializes a ULParameter object with the specified parameter name and data type.</td>
</tr>
<tr>
<td>ULParameter(string, ULDbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object) constructor</td>
<td>Initializes a ULParameter object with the specified parameter name, data type, length, direction, nullability, numeric precision, numeric scale, source column, source version, and value.</td>
</tr>
<tr>
<td>ULParameter(string, ULDbType, int, string) constructor</td>
<td>Initializes a ULParameter object with the specified parameter name, data type, and length.</td>
</tr>
</tbody>
</table>
ULParameter() constructor

Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.

Visual Basic syntax

```vbnet
Public Sub New()
```

C# syntax

```csharp
public ULParameter()
```

See also

- “ULParameter.Value property [UltraLite.NET]” on page 321
- “ULParameter.ULParameter constructor [UltraLite.NET]” on page 309
- “ULCommand class [UltraLite.NET]” on page 71

Example

The following code creates a ULParameter value of 3 and adds it to a ULCommand object named cmd.

```vbnet
' Visual Basic
Dim p As ULParameter = New ULParameter
p.Value = 3
cmd.Parameters.Add( p )
```

The following code is the C# language equivalent:

```csharp
// C#
ULParameter p = new ULParameter();
p.Value = 3;
cmd.Parameters.Add( p );
```

ULParameter(string, object) constructor

Initializes a ULParameter object with the specified parameter name and value.

Visual Basic syntax

```vbnet
Public Sub New(ByVal parameterName As String, ByVal value As Object)
```

C# syntax

```csharp
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 the ULCommand object.

- **value**  Pass a System.Object class to produce 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 the object type when using this constructor.

See also
- “ULParameter.ULParameter constructor [UltraLite.NET]” on page 309
- “ULCommand class [UltraLite.NET]” on page 71
- System.Object

Example
The following code creates a ULParameter value of 0 and adds it to a ULCommand object named cmd.

' Visual Basic
    cmd.Parameters.Add( New ULParameter( "", CType( 0, Object ) ) )

The following code is the C# language equivalent:

    // C#
    cmd.Parameters.Add( new ULParameter( "", (object)0 ) );

ULParameter(string, ULDbType) constructor
Initializes a ULParameter object with the specified parameter name and data type.

Visual Basic syntax
    Public Sub New(ByVal parameterName As String, ByVal dbType As ULDbType)

C# syntax
    public ULParameter(string parameterName, ULDbType dbType)

Parameters
- **parameterName**    The name of the parameter. For unnamed parameters, use an empty string (""") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by the ULCommand object.

- **dbType**    One of the iAnywhere.Data.UltraLite.ULDbType values.

Remarks
This constructor is not recommended; it is provided for compatibility with other data providers.

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.
ULParameter(string, ULDDbType, int) constructor

Initializes a ULParameter object with the specified parameter name and data type.

Visual Basic syntax

```vbnet
Public Sub New(
    ByVal parameterName As String,
    ByVal dbType As ULDDbType,
    ByVal size As Integer)
```

C# syntax

```csharp
public ULParameter(string parameterName, ULDDbType 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 the ULCommand object.
- **dbType**  One of the iAnywhere.Data.UltraLite.ULDbType values.
- **size**  The length of the parameter.

Remarks

This constructor is not recommended; it is provided for compatibility with other data providers.

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

See also

- “ULParameter.ULParameter constructor [UltraLite.NET]” on page 309
- “ULParameter.Value property [UltraLite.NET]” on page 321

ULParameter(string, ULDDbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object) constructor

Initializes a ULParameter object with the specified parameter name, data type, length, direction, nullability, numeric precision, numeric scale, source column, source version, and value.
Visual Basic syntax

Public Sub New(
    ByVal parameterName As String,
    ByVal dbType As ULDbType,
    ByVal size As Integer,
    ByVal direction As ParameterDirection,
    ByVal isNullable As Boolean,
    ByVal precision As Byte,
    ByVal scale As Byte,
    ByVal sourceColumn As String,
    ByVal sourceVersion As DataRowVersion,
    ByVal value As Object
)

C# syntax

public ULParameter(
    string parameterName,
    ULDbType dbType,
    int size,
    ParameterDirection direction,
    bool isNullable,
    byte precision,
    byte scale,
    string sourceColumn,
    DataRowVersion sourceVersion,
    object value
)

Parameters

- **parameterName**  The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by the ULCommand object.

- **dbType**  One of the iAnywhere.Data.UltraLite.ULDbType values.

- **size**  The length of the parameter.

- **direction**  One of the System.Data.ParameterDirection values.

- **isNullable**  True if the value of the field can be null; otherwise, false.

- **precision**  The total number of digits to the left and right of the decimal point to which the Value property is resolved.

- **scale**  The total number of decimal places to which the Value property is resolved.

- **sourceColumn**  The name of the source column to map.


- **value**  Pass a System.Object to produce the value of the parameter.
Exceptions

- **ULException class** Only the System.Data ParameterDirection.Input direction is supported in UltraLite.NET.

Remarks

This constructor is not recommended; it is provided for compatibility with other data providers.

See also

- “ULParameter.ULParameter constructor [UltraLite.NET]” on page 309
- “ULCommand class [UltraLite.NET]” on page 71
- System.Data.DataRowVersion
- System.Object
- System.Data ParameterDirection

**ULParameter(string, ULDbType, int, string) constructor**

Initializes a ULParameter object with the specified parameter name, data type, and length.

Visual Basic syntax

```vbnet
Public Sub New(
    ByVal parameterName As String,
    ByVal dbType As ULDbType,
    ByVal size As Integer,
    ByVal sourceColumn As String
)
```

C# syntax

```csharp
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 the ULCommand object.

- **dbType** One of the iAnywhere.Data.UltraLite.ULDbType values.

- **size** The length of the parameter.

- **sourceColumn** The name of the source column to map.

Remarks

This constructor is not recommended; it is provided for compatibility with other data providers.
In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

See also

- “ULParameter.ULParameter constructor [UltraLite.NET]” on page 309
- “ULParameter.Value property [UltraLite.NET]” on page 321
- “ULCommand class [UltraLite.NET]” on page 71

**ResetDbType method**

This method is not supported in UltraLite.NET.

**Visual Basic syntax**

```vbnet
Public Overrides Sub ResetDbType()
```

**C# syntax**

```csharp
public override void ResetDbType()
```

**Remarks**

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

See also

- “ULParameter.Value property [UltraLite.NET]” on page 321

**ToString method**

Returns the string representation of this instance.

**Visual Basic syntax**

```vbnet
Public Overrides Function ToString() As String
```

**C# syntax**

```csharp
public override string ToString()
```

**Returns**

The name of the parameter.

**DbType property**

Specifies the System.Data.DbType for the parameter.

**Visual Basic syntax**

```vbnet
Public Overrides Property DbType As DbType
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 315
C# syntax

```csharp
public override DbType DbType {get;set;}
```

Exceptions

- **ArgumentException** There is no mapping from the specified value to an iAnywhere.Data.UltraLite.ULDbType value, hence, the specified value is not supported.

Remarks

One of the System.Data.DbType values.

The ULParameter.ULDbType and DbType properties are linked. Therefore, setting the DbType property changes the ULParameter.ULDbType property to a supporting iAnywhere.Data.UltraLite.ULDbType value.

See also

- “ULParameter.ULDbType property [UltraLite.NET]” on page 320
- System.Data.DbType

**Direction property**

A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

Visual Basic syntax

```vbnet
Public Overrides Property Direction As ParameterDirection
```

C# syntax

```csharp
public override ParameterDirection Direction {get;set;}
```

Exceptions

- **ULException class** Only the System.Data.ParameterDirection.Input direction is supported in UltraLite.NET.

Remarks

One of the System.Data.ParameterDirection values.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

See also

- “ULParameter.Value property [UltraLite.NET]” on page 321
- System.Data.ParameterDirection
IsNullable property

Specifies whether the parameter accepts null values.

Visual Basic syntax
Public Overrides Property IsNullable As Boolean

C# syntax
public override bool IsNullable {get;set;}

Remarks
True if null values are accepted, false otherwise. The default is false. Null values are handled using the DBNull class.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

See also
● “ULParameter.Value property [UltraLite.NET]” on page 321

Offset property

Specifies the offset to the ULParameter.Value.

Visual Basic syntax
Public Property Offset As Integer

C# syntax
public int Offset {get;set;}

Remarks
The offset to the value. The default is 0.

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

See also
● “ULParameter.Value property [UltraLite.NET]” on page 321

ParameterName property

Specifies the name of the parameter.

Visual Basic syntax
Public Overrides Property ParameterName As String
C# syntax

```csharp
public override string ParameterName {get;set;}
```

Remarks

A string representing the name of the parameter, or an empty string (""") for unnamed parameters. Specifying a null reference (Nothing in Visual Basic) results in an empty string being used.

In UltraLite.NET, parameter names are not used by an ULCommand object. All parameters are treated as positional parameters and are used by a command in the order that they were added.

See also

- “ULCommand class [UltraLite.NET]” on page 71

**Precision property**

Specifies the maximum number of digits used to represent the ULParameter.Value property.

Visual Basic syntax

```vbnet
Public Property Precision As Byte
```

C# syntax

```csharp
public byte Precision {get;set;}
```

Exceptions

- ArgumentException   The value is greater than 38.

Remarks

The maximum number of digits used to represent the ULParameter.Value property. The default value is 0, which indicates that the data provider sets the precision for the ULParameter.Value property.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

See also

- “ULParameter.Value property [UltraLite.NET]” on page 321

**Scale property**

Specifies the number of decimal places to which ULParameter.Value property is resolved.

Visual Basic syntax

```vbnet
Public Property Scale As Byte
```

C# syntax

```csharp
public byte Scale {get;set;}
```
Remarks

The number of decimal places to which ULParameter.Value property is resolved. The default is 0.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

See also

● “ULParameter.Value property [UltraLite.NET]” on page 321

Size property

Specifies the maximum size of the data within the column.

Visual Basic syntax

Public Overrides Property Size As Integer

C# syntax

global override int Size {get;set;}

Remarks

The maximum size of the data within the column. The default value is inferred from the parameter value. The Size property is used for binary and string types.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

See also

● “ULParameter.Value property [UltraLite.NET]” on page 321

SourceColumn property

Specifies the name of the source column mapped to the DataSet object and used for loading or returning the value.

Visual Basic syntax

Public Overrides Property SourceColumn As String

C# syntax

public override string SourceColumn {get;set;}

Remarks

A string specifying the name of the source column mapped to the DataSet object and used for loading or returning the value.
In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

**See also**
- “ULParameter.Value property [UltraLite.NET]” on page 321

**SourceColumnNullMapping property**
Specifies whether the source column is nullable.

**Visual Basic syntax**
```vbnet
Public Overrides Property SourceColumnNullMapping As Boolean
```

**C# syntax**
```csharp
public override bool SourceColumnNullMapping {get;set;}
```

**Remarks**
True if the source column is nullable; false, otherwise.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

**See also**
- “ULParameter.Value property [UltraLite.NET]” on page 321

**SourceVersion property**
The System.Data.DataRowVersion value to use when loading the ULParameter.Value property.

**Visual Basic syntax**
```vbnet
Public Overrides Property SourceVersion As DataRowVersion
```

**C# syntax**
```csharp
public override DataRowVersion SourceVersion {get;set;}
```

**See also**
- “ULParameter.Value property [UltraLite.NET]” on page 321
  - System.Data.DataRowVersion

**ULDbType property**
Specifies the iAnywhere.Data.UltraLite.ULDbType for the parameter.
Visual Basic syntax

Public Property ULDbType As ULDbType

C# syntax

public ULDbType ULDbType {get;set;}

Remarks

One of the iAnywhere.Data.UltraLite.ULDbType values.

The ULDbType and ULParameter.DbType properties are linked. Therefore, setting the ULDbType property changes the ULParameter.DbType property to a supporting System.Data.DbType value.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the ULParameter.Value property is important.

See also

● “ULParameter.Value property [UltraLite.NET]” on page 321
● “ULParameter.DbType property [UltraLite.NET]” on page 315
● System.Data.DbType

Value property

Specifies the value of the parameter.

Visual Basic syntax

Public Overrides Property Value As Object

C# syntax

public override object Value {get;set;}

Remarks

Pass a System.Object class that specifies the value of the parameter.

The value is sent as-is to the data provider without any type conversion or mapping. When the command is executed, the command attempts to convert the value to the required type, signaling a ULException object with ULSQLCode.SQLE_CONVERSION_ERROR if it cannot convert the value.

See also

● “ULException class [UltraLite.NET]” on page 265
● System.Object

ULParameterCollection class

Represents all parameters to a ULCommand object.
Visual Basic syntax

Public NotInheritable Class ULParameterCollection
End Class

C# syntax

public sealed class ULParameterCollection :

Base classes


Members

All members of the ULParameterCollection class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add method</td>
<td>Adds a ULParameter object to the collection.</td>
</tr>
<tr>
<td>AddRange method</td>
<td>Adds an array of values to the end of the ULParameterCollection.</td>
</tr>
<tr>
<td>Clear method</td>
<td>Removes all the parameters from the collection.</td>
</tr>
<tr>
<td>Contains method</td>
<td>Checks whether a ULParameter object exists in the collection.</td>
</tr>
<tr>
<td>CopyTo method</td>
<td>Copies ULParameter objects from the ULParameterCollection to the specified array.</td>
</tr>
<tr>
<td>GetEnumerator method</td>
<td>Returns an enumerator for the collection.</td>
</tr>
<tr>
<td>IndexOf method</td>
<td>Returns the location of the ULParameter object in the collection.</td>
</tr>
<tr>
<td>Insert method</td>
<td>Inserts an ULParameter object in the collection at the specified index.</td>
</tr>
<tr>
<td>Remove method</td>
<td>Removes an ULParameter object from the collection.</td>
</tr>
<tr>
<td>RemoveAt method</td>
<td>Removes the parameter at the specified index in the collection.</td>
</tr>
<tr>
<td>Count property</td>
<td>Returns the number of ULParameter objects in the collection.</td>
</tr>
</tbody>
</table>
IsFixedSize property
Indicates whether the ULParameterCollection object has a fixed size.

IsReadOnly property
Indicates whether the ULParameterCollection object is read-only.

IsSynchronized property
Indicates whether the ULParameterCollection object is synchronized.

SyncRoot property
Returns an object that can be used to synchronize access to the SA-
ParameterCollection object.

this property
Returns the ULParameter object at the specified index.

Remarks
All parameters in the collection are treated as positional parameters and are specified in the same order as
the question mark placeholders in the ULCommand.CommandText value. For example, the first
parameter in the collection corresponds to the first question mark in the SQL statement, the second
parameter in the collection corresponds to the second question mark in the SQL statement, and so on.
There must be at least as many question marks in the ULCommand.CommandText value as there are
parameters in the collection. Nulls are substituted for missing parameters.

There is no constructor for the ULParameterCollection class. You obtain a ULParameterCollection object
from the ULCommand.Parameters property.

See also
● “ULCommand class [UltraLite.NET]” on page 71
● “ULCommand.CommandText property [UltraLite.NET]” on page 101
● “ULCommand.Parameters property [UltraLite.NET]” on page 105
● System.Data.IDataParameterCollection

Add method
Adds a ULParameter object to the collection.

Overload list
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add(object) method</td>
<td>Adds a ULParameter object to the collection.</td>
</tr>
<tr>
<td>Add(string, object) method</td>
<td>Adds a new ULParameter object, created using the specified parameter name and value, to the collection.</td>
</tr>
<tr>
<td>Add(string, ULDbType) method</td>
<td>Adds a new ULParameter object, created using the specified parameter name and data type, to the collection.</td>
</tr>
</tbody>
</table>
Add(string, ULDbType, int) method
Adds a new ULParameter object, created using the specified parameter name, data type, and length, to the collection.

Add(string, ULDbType, int, string) method
Adds a new ULParameter object, created using the specified parameter name, data type, length, and source column name, to the collection.

Add(ULParameter) method
Adds a ULParameter object to the collection.

Add(object) method
Adds a ULParameter object to the collection.

Visual Basic syntax
Public Overrides Function Add(ByVal value As Object) As Integer

C# syntax
public override int Add(object value)

Parameters
- value The ULParameter object to add to the collection.

Returns
The index of the new ULParameter object.

Exceptions
- ArgumentException The value cannot be null (Nothing in Visual Basic).
- InvalidCastException The value specified must be a ULParameter object.
- ArgumentNullException The ULParameter object can only be added to the collection once.

Remarks
All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText value. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText value as there are parameters in the collection. Nulls are substituted for missing parameters.
Add(string, object) method

Adds a new ULParameter object, created using the specified parameter name and value, to the collection.

Visual Basic syntax

Public Function Add(  
    ByVal parameterName As String,  
    ByVal value As Object  
) As ULParameter

C# syntax

public ULParameter Add(string parameterName, object value)

Parameters

- **parameterName**  The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by the ULCommand object.

- **value**  A System.Object that is to be the value of the parameter.

Returns

The new ULParameter object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText value. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText value 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 the object type when using this method.

See also

- “ULParameterCollection.Add method [UltraLite.NET]” on page 323
- “ULParameter class [UltraLite.NET]” on page 307
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand class [UltraLite.NET]” on page 71
- System.Object
Example

The following code adds a ULParameter value of 0 to a ULCommand object named cmd.

```vbs
' Visual Basic
cmd.Parameters.Add( "", CType( 0, Object ) )
```

The following code is the C# language equivalent:

```csharp
// C#
cmd.Parameters.Add( "", (object)0 );
```

Add(string, ULDbType) method

Adds a new ULParameter object, created using the specified parameter name and data type, to the collection.

**Visual Basic syntax**

```vbnet
Public Function Add(ByVal parameterName As String, ByVal ulDbType As ULDbType) As ULParameter
```

**C# syntax**

```csharp
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 the ULCommand object.

- **ulDbType** One of the iAnywhere.Data.UltraLite.ULDbType values.

**Returns**

The new ULParameter object.

**Remarks**

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText value. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText value as there are parameters in the collection. Nulls are substituted for missing parameters.

**See also**

- “ULParameterCollection.Add method [UltraLite.NET]” on page 323
- “ULParameter class [UltraLite.NET]” on page 307
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand class [UltraLite.NET]” on page 71
Add(string, ULDbType, int) method

Adds a new ULParameter object, created using the specified parameter name, data type, and length, to the collection.

Visual Basic syntax

Public Function Add(ByVal parameterName As String, ByVal ulDbType As ULDbType, ByVal size As Integer) As ULParameter

C# syntax

public ULParameter Add(string parameterName, ULDbType ulDbType, int size)

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by the ULCommand object.

- **ulDbType** One of the iAnywhere.Data.UltraLite.ULDbType values.

- **size** The length of the parameter.

Returns

The new ULParameter object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText value. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText value as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- “ULParameterCollection.Add method [UltraLite.NET]” on page 323
- “ULParameter class [UltraLite.NET]” on page 307
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand class [UltraLite.NET]” on page 71
Add(string, ULDbType, int, string) method

Adds a new ULParameter object, created using the specified parameter name, data type, length, and source column name, to the collection.

Visual Basic syntax

```vbnet
Public Function Add(
    ByVal parameterName As String,
    ByVal ulDbType As ULDbType,
    ByVal size As Integer,
    ByVal sourceColumn As String
) As ULParameter
```

C# syntax

```csharp
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 the ULCommand object.

- **ulDbType**  One of the iAnywhere.Data.UltraLite.ULDbType values.

- **size**  The length of the parameter.

- **sourceColumn**  The name of the source column to map.

Returns

The new ULParameter object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText value. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText value as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- “ULParameterCollection.Add method [UltraLite.NET]” on page 323
- “ULParameter class [UltraLite.NET]” on page 307
- “ULCommand.CommandText property [UltraLite.NET]” on page 101
- “ULCommand class [UltraLite.NET]” on page 71
Add(ULParameter) method

Adds a ULParameter object to the collection.

Visual Basic syntax

Public Function Add(ByVal value As ULParameter) As ULParameter

C# syntax

public ULParameter Add(ULParameter value)

Parameters

● value  The ULParameter object to add to the collection.

Returns

The new ULParameter object.

Exceptions

● ArgumentNullException  The value cannot be null (Nothing in Visual Basic).

● ArgumentException  The ULParameter object can only be added to the collection once.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the ULCommand.CommandText value. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the ULCommand.CommandText value as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

● “ULParameterCollection.Add method [UltraLite.NET]” on page 323
● “ULParameter class [UltraLite.NET]” on page 307
● “ULCommand.CommandText property [UltraLite.NET]” on page 101

AddRange method

Adds an array of values to the end of the ULParameterCollection.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRange(Array) method</td>
<td>Adds an array of values to the end of the ULParameterCollection.</td>
</tr>
<tr>
<td>AddRange(ULParameter[]) method</td>
<td>Adds an array of values to the end of the ULParameterCollection.</td>
</tr>
</tbody>
</table>
AddRange(Array) method

Adds an array of values to the end of the ULParameterCollection.

Visual Basic syntax

Public Overrides Sub AddRange(ByVal values As Array)

C# syntax

public override void AddRange(Array values)

Parameters

● values An array of ULParameter objects to add to the end of this collection.

See also

● “ULParameter class [UltraLite.NET]” on page 307

AddRange(ULParameter[]) method

Adds an array of values to the end of the ULParameterCollection.

Visual Basic syntax

Public Sub AddRange(ByVal values As ULParameter())

C# syntax

public void AddRange(ULParameter[] values)

Parameters

● values An array of ULParameter objects to add to the end of this collection.

Remarks

This is the strongly-typed version of the DbParameterCollection.AddRange(Array) method.

See also

● “ULParameter class [UltraLite.NET]” on page 307
● “ULParameterCollection class [UltraLite.NET]” on page 321

Clear method

Removes all the parameters from the collection.

Visual Basic syntax

Public Overrides Sub Clear()

C# syntax

public override void Clear()
Contains method

Checks whether a ULParameter object exists in the collection.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains(object) method</td>
<td>Checks whether a ULParameter object exists in the collection.</td>
</tr>
<tr>
<td>Contains(string) method</td>
<td>Checks whether a ULParameter object with the specified name exists in the collection.</td>
</tr>
</tbody>
</table>

Contains(object) method

Checks whether a ULParameter object exists in the collection.

Visual Basic syntax

```vbnet
Public Overrides Function Contains(ByVal value As Object) As Boolean
```

C# syntax

```csharp
public override bool Contains(object value)
```

Parameters

- **value** The ULParameter object to check for.

Returns

True if the collection contains the ULParameter object, false otherwise.

See also

- “ULParameterCollection.Contains method [UltraLite.NET]” on page 331
- “ULParameter class [UltraLite.NET]” on page 307

Contains(string) method

Checks whether a ULParameter object with the specified name exists in the collection.

Visual Basic syntax

```vbnet
Public Overrides Function Contains(ByVal value As String) As Boolean
```

C# syntax

```csharp
public override bool Contains(string value)
```

Parameters

- **value** The name of the parameter to search for.
Returns

True if the collection contains the ULParameter object, false otherwise.

See also

- “ULParameterCollection.Contains method [UltraLite.NET]” on page 331
- “ULParameter class [UltraLite.NET]” on page 307

**CopyTo method**

Copies ULParameter objects from the ULParameterCollection to the specified array.

**Visual Basic syntax**

```vbnet
Public Overrides Sub CopyTo(
    ByVal array As Array,
    ByVal index As Integer
)
```

**C# syntax**

```csharp
public override void CopyTo(Array array, int index)
```

**Parameters**

- **array** The array into which to copy the ULParameter objects.
- **index** The starting index of the array.

See also

- “ULParameter class [UltraLite.NET]” on page 307

**GetEnumerator method**

Returns an enumerator for the collection.

**Visual Basic syntax**

```vbnet
Public Overrides Function GetEnumerator() As System.Collections.IEnumerator
```

**C# syntax**

```csharp
public override IEnumerator GetEnumerator()
```

**Returns**

An ArrayList enumerator enumerating the parameters in the collection.
IndexOf method

Returns the location of the ULParameter object in the collection.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.IndexOf(object) method</td>
<td>Returns the location of the ULParameter object in the collection.</td>
</tr>
<tr>
<td>.IndexOf(string) method</td>
<td>Returns the location of the ULParameter object with the specified name in the collection.</td>
</tr>
</tbody>
</table>

IndexOf(object) method

Returns the location of the ULParameter object in the collection.

Visual Basic syntax

```vbnet
Public Overrides Function IndexOf(ByVal value As Object) As Integer
```

C# syntax

```csharp
public override int IndexOf(object value)
```

Parameters

- **value** The ULParameter object to locate.

Returns

The zero-based index of the ULParameter object in the collection or -1 if the parameter is not found.

Exceptions

- **InvalidCastException** The value specified must be a ULParameter object.

See also

- “ULParameterCollection.IndexOf method [UltraLite.NET]” on page 333
- “ULParameter class [UltraLite.NET]” on page 307

IndexOf(string) method

Returns the location of the ULParameter object with the specified name in the collection.

Visual Basic syntax

```vbnet
Public Overrides Function IndexOf(ByVal parameterName As String) As Integer
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 333
C# syntax
   public override int IndexOf(string parameterName)

Parameters
   ● parameterName   The name of the parameter to locate.

Returns
   The zero-based index of the ULParameter object in the collection or -1 if the parameter is not found.

See also
   ● “ULParameterCollection.IndexOf method [UltraLite.NET]” on page 333
   ● “ULParameter class [UltraLite.NET]” on page 307

Insert method
   Inserts an ULParameter object in the collection at the specified index.

Visual Basic syntax
   Public Overrides Sub Insert(
   ByVal index As Integer,
   ByVal value As Object
   )

C# syntax
   public override void Insert(int index, object value)

Parameters
   ● index   The zero-based index where the parameter is to be inserted within the collection.

   ● value   The ULParameter object to insert.

Exceptions
   ● IndexOutOfRangeException   The index is invalid.


   ● InvalidCastException   The value specified must be a ULParameter object.

See also
   ● “ULParameter class [UltraLite.NET]” on page 307

Remove method
   Removes an ULParameter object from the collection.
Visual Basic syntax

Public Overrides Sub Remove(ByVal value As Object)

C# syntax

public override void Remove(object value)

Parameters

● value The ULParameter object to remove.

Exceptions


● InvalidCastException The value specified must be a ULParameter object.

● ArgumentException The collection does not contain the specified parameter.

See also

● “ULParameter class [UltraLite.NET]” on page 307

RemoveAt method

Removes the parameter at the specified index in the collection.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoveAt(int) method</td>
<td>Removes the parameter at the specified index in the collection.</td>
</tr>
<tr>
<td>RemoveAt(string) method</td>
<td>Removes the parameter with the specified name from the collection.</td>
</tr>
</tbody>
</table>

RemoveAt(int) method

Removes the parameter at the specified index in the collection.

Visual Basic syntax

Public Overrides Sub RemoveAt(ByVal index As Integer)

C# syntax

public override void RemoveAt(int index)

Parameters

● index The zero-based index of the parameter to remove. The value must be in the range [0,ULParameterCollection.Count-1]. The first parameter in the collection has an index value of zero.
Exceptions

- **IndexOutOfRangeException**  The index is invalid.

See also

- “ULParameterCollection.RemoveAt method [UltraLite.NET]” on page 335
- “ULParameterCollection.Count property [UltraLite.NET]” on page 336

**RemoveAt(string) method**

Removes the parameter with the specified name from the collection.

**Visual Basic syntax**

```vbnet
Public Overrides Sub RemoveAt(ByVal parameterName As String)
```

**C# syntax**

```csharp
public override void RemoveAt(string parameterName)
```

**Parameters**

- **parameterName**  The name of the parameter to retrieve.

**Exceptions**

- **IndexOutOfRangeException**  There is no parameter with the specified name.

See also

- “ULParameterCollection.RemoveAt method [UltraLite.NET]” on page 335

**Count property**

Returns the number of ULParameter objects in the collection.

**Visual Basic syntax**

```vbnet
Public ReadOnly Overrides Property Count As Integer
```

**C# syntax**

```csharp
public override int Count {get;}
```

**Remarks**

The number of ULParameter objects in the collection.

**IsFixedSize property**

Indicates whether the ULParameterCollection object has a fixed size.
Visual Basic syntax
    Public ReadOnly Overrides Property IsFixedSize As Boolean

C# syntax
    public override bool IsFixedSize {get;}

Remarks
    True if this collection has a fixed size, false otherwise.

**IsReadOnly property**
    Indicates whether the ULParameterCollection object is read-only.

Visual Basic syntax
    Public ReadOnly Overrides Property IsReadOnly As Boolean

C# syntax
    public override bool IsReadOnly {get;}

Remarks
    True if this collection is read-only, false otherwise.

**IsSynchronized property**
    Indicates whether the ULParameterCollection object is synchronized.

Visual Basic syntax
    Public ReadOnly Overrides Property IsSynchronized As Boolean

C# syntax
    public override bool IsSynchronized {get;}

Remarks
    True if this collection is synchronized, false otherwise.

**SyncRoot property**
    Returns an object that can be used to synchronize access to the SAPParameterCollection object.

Visual Basic syntax
    Public ReadOnly Overrides Property SyncRoot As Object

C# syntax
    public override object SyncRoot {get;}
Remarks
The object to be used to synchronize access to this collection.

this property
Returns the ULParameter object at the specified index.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>this[int] property</td>
<td>Returns the ULParameter object at the specified index.</td>
</tr>
<tr>
<td>this[string] property</td>
<td>Returns the ULParameter object with the specified name.</td>
</tr>
</tbody>
</table>

this[int] property
Returns the ULParameter object at the specified index.

Visual Basic syntax
Public Shadows Property Item(ByVal index As Integer) As ULParameter

C# syntax
public new ULParameter this[int index] {get;set;}

Parameters
- index The zero-based index of the parameter to retrieve. The value must be in the range [0,ULParameterCollection.Count-1]. The first parameter in the collection has an index value of zero.

Returns
The ULParameter object at the specified index.

Exceptions
- IndexOutOfRangeException The index is invalid.

Remarks
In C#, this property is the indexer for the ULParameterCollection class.

This is the strongly-typed version of DbParameterCollection.this[int] property.

See also
- “ULParameter class [UltraLite.NET]” on page 307
this[string] property

Returns the ULParameter object with the specified name.

Visual Basic syntax

Public Shadows Property Item(
  ByVal parameterName As String
) As ULParameter

C# syntax

public new ULParameter this[string parameterName] {get;set;}

Parameters

- **parameterName**  The name of the parameter to retrieve.

Returns

The ULParameter object with the specified name.

Exceptions

- **IndexOutOfRangeException**  There is no parameter with the specified name.
- **ArgumentNullException**  You cannot set a parameter using a null (Nothing in Visual Basic) parameter name.

Remarks

In C#, this property is the indexer for the ULParameterCollection class.

This is the strongly-typed version of DbParameterCollection.this[string] property.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.GetValue method [UltraLite.NET]” on page 254
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULParameter class [UltraLite.NET]” on page 307

ULResultSet class

**UL Ext:** Represents an editable result set in an UltraLite database.

Visual Basic syntax

Public Class ULResultSet Inherits ULDataReader

C# syntax

public class ULResultSet : ULDataReader
**Base classes**
- “ULDataReader class [UltraLite.NET]” on page 228

**Derived classes**
- “ULTable class [UltraLite.NET]” on page 401

**Members**
All members of the ULResultSet class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppendBytes method</td>
<td>Appends the specified subset of the specified array of System.Bytes to the new value for the specified ULDbType.LongBinary column.</td>
</tr>
<tr>
<td>AppendChars method</td>
<td>Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType.LongVarchar column.</td>
</tr>
<tr>
<td>Close method</td>
<td>Closes the cursor.</td>
</tr>
<tr>
<td>Delete method</td>
<td>Deletes the current row.</td>
</tr>
<tr>
<td>Dispose method</td>
<td>Releases all resources used by the current instance of the System.Data.Common.DbDataReader class.</td>
</tr>
<tr>
<td>GetBoolean method</td>
<td>Returns the value for the specified column as a System.Boolean.</td>
</tr>
<tr>
<td>GetByte method</td>
<td>Returns the value for the specified column as an unsigned 8-bit value (System.Byte).</td>
</tr>
<tr>
<td>GetBytes method</td>
<td><strong>UL Ext:</strong> Returns the value for the specified column as an array of System.Bytes values.</td>
</tr>
<tr>
<td>GetChar method</td>
<td>This method is not supported in UltraLite.NET.</td>
</tr>
<tr>
<td>GetChars method</td>
<td>Copies a subset of the value for the specified ULDbType.LongVarchar column, beginning at the specified offset, to the specified offset of the destination System.Char array.</td>
</tr>
<tr>
<td>GetDataTypeName method</td>
<td>Returns the name of the specified column's provider data type.</td>
</tr>
<tr>
<td>GetDateTime method</td>
<td>Returns the value for the specified column as a System.DateTime type with millisecond accuracy.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetDecimal method</td>
<td>Returns the value for the specified column as a System.Decimal type.</td>
</tr>
<tr>
<td>GetDouble method</td>
<td>Returns the value for the specified column as a System.Double type.</td>
</tr>
<tr>
<td>GetEnumerator method</td>
<td>Returns an System.Collections.IEnumerator value that iterates through the ULDataReader object.</td>
</tr>
<tr>
<td>GetFieldType method</td>
<td>Returns the System.Type value most appropriate for the specified column.</td>
</tr>
<tr>
<td>GetFieldValue(T) method (Inherited from System.Data.Common.DbDataReader)</td>
<td>Synchronously gets the value of the specified column as a type.</td>
</tr>
<tr>
<td>GetFieldValueAsync(T) method (Inherited from System.Data.Common.DbDataReader)</td>
<td>Asynchronously gets the value of the specified column as a type.</td>
</tr>
<tr>
<td>GetFloat method</td>
<td>Returns the value for the specified column as a System.Single type.</td>
</tr>
<tr>
<td>GetGuid method</td>
<td>Returns the value for the specified column as a UUID (System.Guid) type.</td>
</tr>
<tr>
<td>GetInt16 method</td>
<td>Returns the value for the specified column as a System.Int16 type.</td>
</tr>
<tr>
<td>GetInt32 method</td>
<td>Returns the value for the specified column as a System.Int32 type.</td>
</tr>
<tr>
<td>GetInt64 method</td>
<td>Returns the value for the specified column as a System.Int64 type.</td>
</tr>
<tr>
<td>GetName method</td>
<td>Returns the name of the specified column.</td>
</tr>
<tr>
<td>GetOrdinal method</td>
<td>Returns the column ID of the named column.</td>
</tr>
<tr>
<td>GetProviderSpecificFieldType method (Inherited from System.Data.Com-</td>
<td>Returns the provider-specific field type of the specified column.</td>
</tr>
<tr>
<td>mon.DbDataReader)</td>
<td>Gets the value of the specified column as an instance of System.Object.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetProviderSpecificValues method</td>
<td>Gets all provider-specific attribute columns in the collection for the current row.</td>
</tr>
<tr>
<td>GetRowCount method</td>
<td><strong>UL Ext:</strong> Returns the number of rows in the cursor, within threshold.</td>
</tr>
<tr>
<td>GetSchemaTable method</td>
<td>Returns a System.Data.DataTable value that describes the column metadata of the ULDataReader object.</td>
</tr>
<tr>
<td>GetStream method</td>
<td>Returns data as a System.IO.Stream.</td>
</tr>
<tr>
<td>GetString method</td>
<td>Returns the value for the specified column as a System.String type.</td>
</tr>
<tr>
<td>GetTextReader method</td>
<td>Retrieves data as a System.IO.TextReader.</td>
</tr>
<tr>
<td>GetTimeSpan method</td>
<td>Returns the value for the specified column as a System.TimeSpan type with millisecond accuracy.</td>
</tr>
<tr>
<td>GetUInt16 method</td>
<td>Returns the value for the specified column as a System.UInt16 type.</td>
</tr>
<tr>
<td>GetUInt32 method</td>
<td>Returns the value for the specified column as a System.UInt32 type.</td>
</tr>
<tr>
<td>GetUInt64 method</td>
<td>Returns the value for the specified column as a System.UInt64 type.</td>
</tr>
<tr>
<td>GetValue method</td>
<td>Returns the value of the specified column in its native format.</td>
</tr>
<tr>
<td>GetValues method</td>
<td>Returns all the column values for the current row.</td>
</tr>
<tr>
<td>IsDBNull method</td>
<td>Checks whether the value from the specified column is NULL.</td>
</tr>
<tr>
<td>IsDBNullAsync method</td>
<td>An asynchronous version of System.Data.Common.DbDataReader.IsDBNull(System.Int32), which gets a value that indicates whether the column contains non-existent or missing values.</td>
</tr>
<tr>
<td>MoveAfterLast method</td>
<td><strong>UL Ext:</strong> Positions the cursor to after the last row of the cursor.</td>
</tr>
<tr>
<td>MoveBeforeFirst method</td>
<td><strong>UL Ext:</strong> Positions the cursor to before the first row of the cursor.</td>
</tr>
<tr>
<td>MoveFirst method</td>
<td><strong>UL Ext:</strong> Positions the cursor to the first row of the cursor.</td>
</tr>
<tr>
<td>MoveLast method</td>
<td><strong>UL Ext:</strong> Positions the cursor to the last row of the cursor.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>MoveNext method</strong></td>
<td><strong>UL Ext:</strong> Positions the cursor to the next row or after the last row if the cursor was already on the last row.</td>
</tr>
<tr>
<td><strong>MovePrevious method</strong></td>
<td><strong>UL Ext:</strong> Positions the cursor to the previous row or before the first row.</td>
</tr>
<tr>
<td><strong>MoveRelative method</strong></td>
<td><strong>UL Ext:</strong> Positions the cursor relative to the current row.</td>
</tr>
<tr>
<td><strong>NextResult method</strong></td>
<td>Advances the ULSDataReader object to the next result when reading the results of batch SQL statements.</td>
</tr>
<tr>
<td><strong>Read method</strong></td>
<td>Positions the cursor to the next row, or after the last row if the cursor was already on the last row.</td>
</tr>
<tr>
<td><strong>ReadAsync method</strong></td>
<td>An asynchronous version of System.Data.Common.DbDataReader.Read, which advances the reader to the next record in a result set.</td>
</tr>
<tr>
<td><strong>SetBoolean method</strong></td>
<td>Sets the value for the specified column using a System.Boolean.</td>
</tr>
<tr>
<td><strong>SetByte method</strong></td>
<td>Sets the value for the specified column using a System.Byte (unsigned 8-bit integer).</td>
</tr>
<tr>
<td><strong>SetBytes method</strong></td>
<td>Sets the value for the specified column using an array of System.Bytes.</td>
</tr>
<tr>
<td><strong>SetDateTime method</strong></td>
<td>Sets the value for the specified column using a System.DateTime.</td>
</tr>
<tr>
<td><strong>SetDBNull method</strong></td>
<td>Sets a column to NULL.</td>
</tr>
<tr>
<td><strong>SetDecimal method</strong></td>
<td>Sets the value for the specified column using a System.Decimal.</td>
</tr>
<tr>
<td><strong>SetDouble method</strong></td>
<td>Sets the value for the specified column using a System.Double.</td>
</tr>
<tr>
<td><strong>SetFloat method</strong></td>
<td>Sets the value for the specified column using a System.Single.</td>
</tr>
<tr>
<td><strong>SetGuid method</strong></td>
<td>Sets the value for the specified column using a System.Guid.</td>
</tr>
<tr>
<td><strong>SetInt16 method</strong></td>
<td>Sets the value for the specified column using a System.Int16.</td>
</tr>
<tr>
<td><strong>SetInt32 method</strong></td>
<td>Sets the value for the specified column using a System.Int32.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SetInt64 method</td>
<td>Sets the value for the specified column using an Int64.</td>
</tr>
<tr>
<td>SetString method</td>
<td>Sets the value for the specified column using a System.String.</td>
</tr>
<tr>
<td>SetTimeSpan method</td>
<td>Sets the value for the specified column using a System.TimeSpan.</td>
</tr>
<tr>
<td>SetToDefault method</td>
<td>Sets the value for the specified column to its default value.</td>
</tr>
<tr>
<td>SetUInt16 method</td>
<td>Sets the value for the specified column using a System.UInt16.</td>
</tr>
<tr>
<td>SetUInt32 method</td>
<td>Sets the value for the specified column using a System.UInt32.</td>
</tr>
<tr>
<td>SetUInt64 method</td>
<td>Sets the value for the specified column using a System.UInt64.</td>
</tr>
<tr>
<td>Update method</td>
<td>Updates the current row with the current column values (specified using the set methods).</td>
</tr>
<tr>
<td>UpdateBegin method</td>
<td>Prepares to update the current row.</td>
</tr>
<tr>
<td>Depth property</td>
<td>Returns the depth of nesting for the current row.</td>
</tr>
<tr>
<td>FieldCount property</td>
<td>Returns the number of columns in the cursor.</td>
</tr>
<tr>
<td>HasRows property</td>
<td>Checks whether the ULDataReader object has one or more rows.</td>
</tr>
<tr>
<td>IsBOF property</td>
<td><strong>UL Ext:</strong> Checks whether the current row position is before the first row.</td>
</tr>
<tr>
<td>IsClosed property</td>
<td>Checks whether the cursor is currently open.</td>
</tr>
<tr>
<td>IsEOF property</td>
<td><strong>UL Ext:</strong> Checks whether the current row position is after the last row.</td>
</tr>
<tr>
<td>RecordsAffected property</td>
<td>Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.</td>
</tr>
<tr>
<td>RowCount property</td>
<td><strong>UL Ext:</strong> Returns the number of rows in the cursor.</td>
</tr>
<tr>
<td>Schema property</td>
<td><strong>UL Ext:</strong> Holds the schema of this cursor.</td>
</tr>
<tr>
<td>this property</td>
<td>Returns the value of the specified column in its native format.</td>
</tr>
<tr>
<td>VisibleFieldCount property</td>
<td>Gets the number of fields in the System.Data.Common.DbDataReader that are not hidden.</td>
</tr>
</tbody>
</table>

**UltraLite.NET API reference**

344 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
Remarks

There is no constructor for this class. Result sets are created using the ULCommand.ExecuteResultSet method.

' Visual Basic
Dim cmd As ULCommand = New ULCommand( _
   "SELECT emp_id FROM employee", conn _
) 
Dim resultSet As ULResultSet = cmd.ExecuteResultSet()

The following code is the C# language equivalent:

// C#
ULCommand cmd = new ULCommand(
   "SELECT emp_id FROM employee", conn
);
ULResultSet resultSet = cmd.ExecuteResultSet();

A ULResultSet object represents an editable result set on which you can perform positioned updates and deletes. For fully editable result sets, use the ULCommand.ExecuteTable method or the ULDataAdapter class.

See also

● “ULCommand.ExecuteResultSet method [UltraLite.NET]” on page 95
● “ULCommand class [UltraLite.NET]” on page 71
● “ULResultSet class [UltraLite.NET]” on page 339
● “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
● “ULDataAdapter class [UltraLite.NET]” on page 204
● “ULDataReader class [UltraLite.NET]” on page 228
● System.Data.IDataReader
● System.Data.IDataRecord
● System.IDisposable

AppendBytes method

Appends the specified subset of the specified array of System.Byte to the new value for the specified ULDbType.LongBinary column.

Visual Basic syntax

Public Sub AppendBytes( 
   ByVal colID As Integer, 
   ByVal val As Byte(), 
   ByVal srcOffset As Integer, 
   ByVal count As Integer 
)

C# syntax

public void AppendBytes(int colID, byte[] val, int srcOffset, int count)
Parameters

- **colID**  The ID number of the column. The value must be in the range [0, ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val**  The value to append to the current new value for the column.
- **srcOffset**  The start position in the source array.
- **count**  The number of bytes to be copied.

Exceptions

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

Remarks

The bytes at position `srcOffset` (starting from 0) through `srcOffset + count` - 1 of the array `val` are appended to the value for the specified column.

When inserting, ULTable.InsertBegin initializes the new value to the column's default value. The data in the row is not actually changed until you execute an ULTable.Insert, and changes are not made permanent until 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 ULEException with code ULSQLCode.SQLE_INVALID_PARAMETER is thrown and the destination is not modified:

- **val** is null.
- **srcOffset** is negative.
- **count** is negative.
- **srcOffset + count** is greater than the `val` length.

For other errors, a ULEException with the appropriate error code is thrown.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.InsertBegin method [UltraLite.NET]” on page 416
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULException class [UltraLite.NET]” on page 265
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Byte
AppendChars method

Appends the specified subset of the specified array of System.Char to the new value for the specified ULDbType.LongVarchar column.

Visual Basic syntax

Public Sub AppendChars(
  ByVal colID As Integer,
  ByVal val As Char(),
  ByVal srcOffset As Integer,
  ByVal count As Integer
)

C# syntax

public void AppendChars(int colID, char[] val, int srcOffset, int count)

Parameters

- colID    The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- val      The value to append to the current new value for the column.
- srcOffset    The start position in the source array.
- count      The number of bytes to be copied.

Exceptions

- ULEException class    A SQL error occurred.

Remarks

The characters at position srcOffset (starting from 0) through srcOffset +count -1 of the array val are appended to the value for the specified column. When inserting, ULTable.InsertBegin initializes the new value to the column's default value. The data in the row is not actually changed until you execute an ULTable.Insert, and changes are not made permanent until 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 ULEException with code ULSQLCode.SQLE_INVALID_PARAMETER is thrown and the destination is not modified:

- val is null.
- srcOffset is negative.
- count is negative.
- srcOffset +count is greater than val length.

For other errors, a ULEException with the appropriate error code is thrown.
**Delete method**

Deletes the current row.

**Visual Basic syntax**

```vbnet
Public Sub Delete()
```

**C# syntax**

```csharp
public void Delete()
```

**Exceptions**

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

**See also**

- “ULConnection.StartSynchronizationDelete method [UltraLite.NET]” on page 150
- “ULConnection.StopSynchronizationDelete method [UltraLite.NET]” on page 150

**SetBoolean method**

Sets the value for the specified column using a System.Boolean.

**Visual Basic syntax**

```vbnet
Public Sub SetBoolean(ByVal colID As Integer, ByVal val As Boolean)
```

**C# syntax**

```csharp
public void SetBoolean(int colID, bool val)
```

**Parameters**

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

- **val** The new value for the column.
Exceptions

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

Remarks

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Boolean

**SetByte method**

Sets the value for the specified column using a System.Byte (unsigned 8-bit integer).

**Visual Basic syntax**

```vbnet
Public Sub SetByte(ByVal colID As Integer, ByVal val As Byte)
```

**C# syntax**

```csharp
public void SetByte(int colID, byte val)
```

Parameters

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

- **val**  The new value for the column.

Exceptions

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

Remarks

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.
SetBytes method

Sets the value for the specified column using an array of System.Byte.

Visual Basic syntax

Public Sub SetBytes(ByVal colID As Integer, ByVal val As Byte())

C# syntax

public void SetBytes(int colID, byte[] val)

Parameters

- colID  The ID number of the column. The value must be in the range [0, ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- val    The new value for the column.

Exceptions

- ULException class  A SQL error occurred.

Remarks

Only suitable for columns of type ULDbType.Binary or ULDbType.LongBinary, or for columns of type ULDbType.UniqueIdentifier when the value is of length 16. The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Byte
SetDateTime method

Sets the value for the specified column using a System.DateTime.

Visual Basic syntax

Public Sub SetDateTime(ByVal colID As Integer, ByVal val As Date)

C# syntax

public void SetDateTime(int colID, DateTime val)

Parameters

- **colID**  The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val**  The new value for the column.

Exceptions

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

Remarks

The set value is accurate to the millisecond. The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.DateTime

SetDBNull method

Sets a column to NULL.

Visual Basic syntax

Public Sub SetDBNull(ByVal colID As Integer)

C# syntax

public void SetDBNull(int colID)

Parameters

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

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

Remarks

The data is not actually changed until you execute an ULTable.Insert or Update, and changes are not made permanent until committed.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULTableSchema.IsColumnNullable method [UltraLite.NET]” on page 431
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260

### SetDecimal method

Sets the value for the specified column using a System.Decimal.

**Visual Basic syntax**

```vbnet
Public Sub SetDecimal(ByVal colID As Integer, ByVal val As Decimal)
```

**C# syntax**

```csharp
public void SetDecimal(int colID, decimal val)
```

Parameters

- **colID**  The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val**  The new value for the column.

Exceptions

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

Remarks

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.
SetDouble method

Sets the value for the specified column using a System.Double.

Visual Basic syntax

Public Sub SetDouble(ByVal colID As Integer, ByVal val As Double)

C# syntax

public void SetDouble(int colID, double val)

Parameters

- colID   The ID number of the column. The value must be in the range [0, ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- val     The new value for the column.

Exceptions

- ULException class  A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Decimal

SetFloat method

Sets the value for the specified column using a System.Single.
Visual Basic syntax

Public Sub SetFloat(ByVal colID As Integer, ByVal val As Single)

C# syntax

class public void SetFloat(int colID, float val)

Parameters

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

- val    The new value for the column.

Exceptions

- ArgumentException A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also

- ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- ULDataReader.Schema property [UltraLite.NET]” on page 263
- ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- ULTable.Insert method [UltraLite.NET]” on page 415
- ULResultSet.Update method [UltraLite.NET]” on page 362
- ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Single

SetGuid method

Sets the value for the specified column using a System.Guid.

Visual Basic syntax

Public Sub SetGuid(ByVal colID As Integer, ByVal val As Guid)

C# syntax

public void SetGuid(int colID, Guid val)

Parameters

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

- val    The new value for the column.
Exceptions

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

Remarks

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed. Only valid for columns of type ULDbType.UniqueIdentifier or for columns of type ULDbType.Binary with a length of 16.

See also

- “ULConnection.GetNewUUID method [UltraLite.NET]” on page 140
- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULCursorSchema.GetColumnSize method [UltraLite.NET]” on page 201
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Guid

### SetInt16 method

Sets the value for the specified column using a System.Int16.

**Visual Basic syntax**

```vbnet
Public Sub SetInt16(ByVal colID As Integer, ByVal val As Short)
```

**C# syntax**

```csharp
public void SetInt16(int colID, short val)
```

**Parameters**

- **colID**  The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val**  The new value for the column.

**Exceptions**

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

**Remarks**

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.
See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Int16

**SetInt32 method**

Sets the value for the specified column using a System.Int32.

**Visual Basic syntax**

```visualbasic
Public Sub SetInt32(ByVal colID As Integer, ByVal val As Integer)
```

**C# syntax**

```csharp
public void SetInt32(int colID, int val)
```

**Parameters**

- **colID** The ID number of the column. The value must be in the range [0, ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

**Exceptions**

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

**Remarks**

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Int32

**SetInt64 method**

Sets the value for the specified column using an Int64.
Visual Basic syntax

Public Sub SetInt64(ByVal colID As Integer, ByVal val As Long)

C# syntax

public void SetInt64(int colID, long val)

Parameters

- colID The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- val The new value for the column.

Exceptions

- ULException class A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.Int64

SetString method

Sets the value for the specified column using a System.String.

Visual Basic syntax

Public Sub SetString(ByVal colID As Integer, ByVal val As String)

C# syntax

public void SetString(int colID, string val)

Parameters

- colID The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- val The new value for the column.
Exceptions

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

Remarks

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260

---

SetTimeSpan method

Sets the value for the specified column using a System.TimeSpan.

**Visual Basic syntax**

```vbnet
Public Sub SetTimeSpan(ByVal colID As Integer, ByVal val As TimeSpan)
```

**C# syntax**

```csharp
public void SetTimeSpan(int colID, TimeSpan val)
```

Parameters

- **colID** The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- **val** The new value for the column.

Exceptions

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

Remarks

The set value is accurate to the millisecond and is normalized to a nonnegative value between 0 and 24 hours. The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.
See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260

System.TimeSpan

SetToDefault method

Sets the value for the specified column to its default value.

Visual Basic syntax

Public Sub SetToDefault(ByVal colID As Integer)

C# syntax

public void SetToDefault(int colID)

Parameters

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

Exceptions

- ULException class A SQL error occurred.

Remarks

The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also

- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTableSchema.GetColumnDefaultValue method [UltraLite.NET]” on page 423
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260

SetUInt16 method

Sets the value for the specified column using a System.UInt16.
Visual Basic syntax
    Public Sub SetUInt16(ByVal colID As Integer, ByVal val As UShort)

C# syntax
    public void SetUInt16(int colID, ushort val)

Parameters
    ● colID    The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
    ● val      The new value for the column.

Exceptions
    ● ULException class    A SQL error occurred.

Remarks
    The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also
    ● “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
    ● “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
    ● “ULTable.Insert method [UltraLite.NET]” on page 415
    ● “ULResultSet.Update method [UltraLite.NET]” on page 362
    ● “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
    ● System.UInt16

SetUInt32 method
    Sets the value for the specified column using a System.UInt32.

Visual Basic syntax
    Public Sub SetUInt32(ByVal colID As Integer, ByVal val As UInteger)

C# syntax
    public void SetUInt32(int colID, uint val)

Parameters
    ● colID    The ID number of the column. The value must be in the range [0,ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
    ● val      The new value for the column.

Exceptions
    ● ULException class    A SQL error occurred.
Remarks
The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also
- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.UInt32

SetUInt64 method
Sets the value for the specified column using a System.UInt64.

Visual Basic syntax
Public Sub SetUInt64(ByVal colID As Integer, ByVal val As ULong)

C# syntax
public void SetUInt64(int colID, ulong val)

Parameters
- colID The ID number of the column. The value must be in the range [0, ULDataReader.FieldCount-1]. The first column in the cursor has an ID value of zero.
- val The new value for the column.

Exceptions
- ULException class A SQL error occurred.

Remarks
The data in the row is not actually changed until you execute a ULTable.Insert or Update method, and changes are not made permanent until committed.

See also
- “ULDataReader.GetOrdinal method [UltraLite.NET]” on page 247
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULTable.Insert method [UltraLite.NET]” on page 415
- “ULResultSet.Update method [UltraLite.NET]” on page 362
- “ULDataReader.FieldCount property [UltraLite.NET]” on page 260
- System.UInt64
**Update method**

Updates the current row with the current column values (specified using the set methods).

**Visual Basic syntax**

```vbnet
Public Sub Update()
```

**C# syntax**

```csharp
public void Update()
```

**Exceptions**

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

**See also**

- “ULResultSet.UpdateBegin method [UltraLite.NET]” on page 362

---

**UpdateBegin method**

Prepares to update the current row.

**Visual Basic syntax**

```vbnet
Public Sub UpdateBegin()
```

**C# syntax**

```csharp
public void UpdateBegin()
```

**Exceptions**

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

**Remarks**

Column values are modified by calling the appropriate setType or AppendType method(s). The first append on a column clears the current column value prior to appending the new value.

The data in the row is not actually changed until you call the Update method, and changes are not made permanent until committed.

Modifying columns in the index used to open the table affects active searches in unpredictable ways. Columns in the primary key of the table can not be updated.

**See also**

- “ULResultSet.Update method [UltraLite.NET]” on page 362

---

**ULResultSetSchema class**

**UL Ext:** Represents the schema of an UltraLite result set.
Visual Basic syntax
Public NotInheritable Class ULResultSetSchema Inherits ULCursorSchema

C# syntax
public sealed class ULResultSetSchema : ULCursorSchema

Base classes
● “ULCursorSchema class [UltraLite.NET]” on page 197

Members
All members of the ULResultSetSchema class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetColumnID method</td>
<td>Returns the column ID of the named column.</td>
</tr>
<tr>
<td>GetColumnName method</td>
<td>Returns the name of the column identified by the specified column ID.</td>
</tr>
<tr>
<td>GetColumnPrecision method</td>
<td>Returns the precision of the column identified by the specified column ID if the column is a numeric column (the NUMERIC SQL type).</td>
</tr>
<tr>
<td>GetColumnScale method</td>
<td>Returns the scale of the column identified by the specified column ID if the column is a numeric column (the NUMERIC SQL type).</td>
</tr>
<tr>
<td>GetColumnSize method</td>
<td>Returns the size of the column identified by the specified column ID if the column is a sized column (the BINARY or CHAR SQL types).</td>
</tr>
<tr>
<td>GetColumnType method</td>
<td>Returns the UltraLite.NET data type of the column identified by the specified column ID.</td>
</tr>
<tr>
<td>GetSchemaTable method</td>
<td>Returns a System.Data.DataTable that describes the column schema of the ULDataReader object.</td>
</tr>
<tr>
<td>ColumnCount property</td>
<td>Returns the number of columns in the cursor.</td>
</tr>
<tr>
<td>IsOpen property</td>
<td>Checks whether the cursor schema is currently open.</td>
</tr>
<tr>
<td>Name property</td>
<td>Returns the name of the cursor.</td>
</tr>
</tbody>
</table>

Remarks
There is no constructor for this class. A ULResultSetSchema object is attached to a result set as its ULDataReader.Schema property.

A result set schema is only valid while the data reader is open.
See also

- “ULCommand class [UltraLite.NET]” on page 71
- “ULDataReader class [UltraLite.NET]” on page 228
- “ULResultSetSchema class [UltraLite.NET]” on page 362
- “ULDataReader.Schema property [UltraLite.NET]” on page 263
- “ULCursorSchema class [UltraLite.NET]” on page 197

**Name property**

Returns the name of the cursor.

**Visual Basic syntax**

```vbnet
Public ReadOnly Overrides Property Name As String
```

**C# syntax**

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

**Remarks**

The SQL statement that generated the ULResultSetSchema.

**ULRowsCopiedEventArgs class**

Represents the set of arguments passed to the ULRowsCopiedEventHandler object.

**Visual Basic syntax**

```vbnet
Public NotInheritable Class ULRowsCopiedEventArgs
```

**C# syntax**

```csharp
public sealed class ULRowsCopiedEventArgs
```

**Members**

All members of the ULRowsCopiedEventArgs class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULRowsCopiedEventArgs constructor</td>
<td>Creates a new instance of the ULRowsCopiedEventArgs object.</td>
</tr>
<tr>
<td>Abort property</td>
<td>Gets or sets a value that indicates whether the bulk-copy operation should be aborted.</td>
</tr>
<tr>
<td>RowsCopied property</td>
<td>Returns the number of rows copied during the current bulk-copy operation.</td>
</tr>
</tbody>
</table>
Remarks

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

See also

● “ULRowsCopiedEventHandler delegate [UltraLite.NET]” on page 439

ULRowsCopiedEventArgs constructor

Creates a new instance of the ULRowsCopiedEventArgs object.

Visual Basic syntax

Public Sub New(ByVal rowsCopied As Long)

C# syntax

public ULRowsCopiedEventArgs(long rowsCopied)

Parameters

● rowsCopied  A 64-bit integer value that indicates the number of rows copied during the current bulk-copy operation.

Remarks

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

Abort property

Gets or sets a value that indicates whether the bulk-copy operation should be aborted.

Visual Basic syntax

Public Property Abort As Boolean

C# syntax

public bool Abort {get;set;}

Remarks

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

RowsCopied property

Returns the number of rows copied during the current bulk-copy operation.

Visual Basic syntax

Public ReadOnly Property RowsCopied As Long
C# syntax

```csharp
public long RowsCopied {get;}
```

Remarks

A long integer representing the number of rows copied.

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

ULRowUpdatedEventArgs class

Provides data for the ULDataAdapter.RowUpdated event.

Visual Basic syntax

```vbnet
Public NotInheritable Class ULRowUpdatedEventArgs
End Class
```

C# syntax

```csharp
```

Base classes


Members

All members of the ULRowUpdatedEventArgs class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| ULRowUpdatedEventArgs       | Constructor
ds constructor Initializes a new instance of the ULRowUpdatedEvent- |
<p>| CopyToRows method           | Copies references to the modified rows into the provided array. |
| Command property            | Returns the ULCommand object executed when the DbDataAdapter.Update method is called. |
| RecordsAffected property    | Returns the number of rows changed, inserted, or deleted by the execution of the SQL statement. |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowCount property (Inherited from System.Data.Common.RowUpdatedEventArgs)</td>
<td>Gets the number of rows processed in a batch of updated records.</td>
</tr>
<tr>
<td>StatementType property (Inherited from System.Data.Common.RowUpdatedEventArgs)</td>
<td>Gets the type of SQL statement executed.</td>
</tr>
</tbody>
</table>

**See also**

- “ULDataAdapter.RowUpdated event [UltraLite.NET]” on page 213

### ULRowUpdatedEventArgs constructor

Initializes a new instance of the ULRowUpdatedEventArgs class.

**Visual Basic syntax**

```vbnet
Public Sub New(
    ByVal row As DataRow,
    ByVal command As IDbCommand,
    ByVal statementType As StatementType,
    ByVal tableMapping As DataTableMapping
)
```

**C# syntax**

```csharp
public ULRowUpdatedEventArgs(
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
)
```

**Parameters**

● **command** The System.Data.IDbCommand executed when the DbDataAdapter.Update method is called.

● **statementType** One of the System.Data.StatementType values that specifies the type of query executed.


See also

● System.Data.DataRow
● System.Data.IDbCommand
● System.Data.StatementType
● System.Data.Common.DataTableMapping

**Command property**

Returns the ULCommand object executed when the DbDataAdapter.Update method is called.

**Visual Basic syntax**

```vbnet
Public ReadOnly Shadows Property Command As ULCommand
```

**C# syntax**

```csharp
public new ULCommand Command {get;}
```

Remarks

The ULCommand object executed by the update.

This is the strongly-typed version of System.Data.Common.RowUpdatedEventArgs.Command.

See also

● “ULCommand class [UltraLite.NET]” on page 71

**RecordsAffected property**

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

**Visual Basic syntax**

```vbnet
Public ReadOnly Shadows Property RecordsAffected As Integer
```

**C# syntax**

```csharp
public new int RecordsAffected {get;}
```
Remarks
For SELECT statements this value is -1.

The number of rows changed, inserted, or deleted; 0 if no rows were affected or the statement failed; and -1 for SELECT statements.

ULRowUpdatingEventArgs class
Provides data for the ULDataAdapter.RowUpdating event.

Visual Basic syntax
Public NotInheritable Class ULRowUpdatingEventArgs
End Class

C# syntax

Base classes

Members
All members of the ULRowUpdatingEventArgs class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULRowUpdatingEventArgs constructor</td>
<td>Initializes a new instance of the ULRowUpdatingEventArgs class.</td>
</tr>
<tr>
<td><strong>BaseCommand</strong> property (Inherited from System.Data.Common.RowUpdatingEventArgs)</td>
<td>Gets or sets the System.Data.IDbCommand object for an instance of this class.</td>
</tr>
<tr>
<td><strong>Command</strong> property</td>
<td>Specifies the ULCommand object to execute when performing the DbDataAdapter.Update method.</td>
</tr>
<tr>
<td><strong>Row</strong> property (Inherited from System.Data.Common.RowUpdatingEventArgs)</td>
<td>Gets the System.Data.DataRow that will be sent to the server as part of an insert, update, or delete operation.</td>
</tr>
<tr>
<td><strong>StatementType</strong> property (Inherited from System.Data.Common.RowUpdatingEventArgs)</td>
<td>Gets the type of SQL statement to execute.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>

See also

- “ULDataAdapter.RowUpdating event [UltraLite.NET]” on page 214

**ULRowUpdatingEventArgs constructor**

Initializes a new instance of the ULRowUpdatingEventArgs class.

**Visual Basic syntax**

```vbnet
Public Sub New(
    ByVal row As DataRow,
    ByVal command As IDbCommand,
    ByVal statementType As StatementType,
    ByVal tableMapping As DataTableMapping
)
```

**C# syntax**

```csharp
public ULRowUpdatingEventArgs (  
    DataRow row,  
    IDbCommand command,  
    StatementType statementType,  
    DataTableMapping tableMapping
)
```

**Parameters**

- **row** The System.Data.DataRow to update.
- **command** The System.Data.IDbCommand to execute during the update.
- **statementType** One of the System.Data.StatementType values that specifies the type of query executed.
See also
- System.Data.DataRow
- System.Data.IDbCommand
- System.Data.StatementType
- System.Data.Common.DataTableMapping

Command property
Specifies the ULCommand object to execute when performing the DbDataAdapter.Update method.

Visual Basic syntax
```vbnet
Public Shadows Property Command As ULCommand
```

C# syntax
```csharp
public new ULCommand Command {get;set;}
```

Remarks
The ULCommand object to execute when updating.

This is the strongly-typed version of the System.Data.Common.RowUpdatingEventArgs.Command value.

See also
- “ULCommand class [UltraLite.NET]” on page 71

ULServerSyncListener interface
UL Ext: The listener interface for receiving server synchronization messages.

Visual Basic syntax
```vbnet
Public Interface ULServerSyncListener
```

C# syntax
```csharp
public interface ULServerSyncListener
```

Members
All members of the ULServerSyncListener interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerSyncInvoked method</td>
<td>Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.</td>
</tr>
</tbody>
</table>
**ServerSyncInvoked method**

Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

**Visual Basic syntax**

Public Sub ServerSyncInvoked(ByVal messageName As String)

**C# syntax**

public void ServerSyncInvoked(string messageName)

**Parameters**

- **messageName**  The name of the message sent to the application.

**Remarks**

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

**Example**

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.mysmq", _ "myCompany.myapp", Me _) ' Create Connection ...


Public Sub ServerSyncInvoked(ByVal messageName As String) _ Implements ULServerSyncListener.ServerSyncInvoked

Me.Invoke(New EventHandler(AddressOf Me.ServerSyncAction)) End Sub

Public Sub ServerSyncAction( _ ByVal sender As Object, ByVal e As EventArgs _) ' Do Server sync conn.Synchronize() End Sub End Class

The following C# code demonstrates how to receive a server synchronization request and perform a synchronization in the UI thread.
using iAnywhere.Data.UltraLite;
public class Form1 : System.Windows.Forms.Form, ULServerSyncListener
{
    private System.Windows.Forms.MainMenu mainMenu1;
    private ULConnection conn;

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

        // TODO: Add any constructor code after
        // InitializeComponent call
        //
        ULConnection.DatabaseManager.SetServerSyncListener("myCompany.mymsg", "myCompany.myapp", this);
        // Create connection
        ... 
    }

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

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

    public void ServerSyncInvoked(string messageName)
    {
        this.Invoke(new EventHandler(ServerSyncHandler));
    }

    internal void ServerSyncHandler(object sender, EventArgs e)
    {
        conn.Synchronize();
    }
}

ULSqlProgressData class

UL Ext: Returns SQL passthrough script progress monitoring data.

Visual Basic syntax

Public Class ULSqlProgressData
C# syntax

```csharp
public class ULSqlProgressData

Members

All members of the ULSqlProgressData class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentScript property</td>
<td>The index of the scripts executed so far.</td>
</tr>
<tr>
<td>ScriptCount property</td>
<td>Returns the number of scripts being executed.</td>
</tr>
<tr>
<td>State property</td>
<td>Returns the current progress state.</td>
</tr>
</tbody>
</table>

CurrentScript property

The index of the scripts executed so far.

Visual Basic syntax

```vbnet
Public ReadOnly Property CurrentScript As Long
```

C# syntax

```csharp
public long CurrentScript {get;}
```

Remarks

The current index of the scripts being executed.

ScriptCount property

Returns the number of scripts being executed.

Visual Basic syntax

```vbnet
Public ReadOnly Property ScriptCount As Long
```

C# syntax

```csharp
public long ScriptCount {get;}
```

Remarks

The number of scripts being executed.

State property

Returns the current progress state.
Visual Basic syntax

    Public ReadOnly Property State As ULSqlProgressState

C# syntax

    public ULSqlProgressState State {get;}

Remarks

    One of the ULSqlProgressState values specifying the current SQL passthrough callback state.

See also

    ● “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

ULSyncParms class

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

Visual Basic syntax

    Public NotInheritable Class ULSyncParms

C# syntax

    public sealed class ULSyncParms

Members

    All members of the ULSyncParms class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopyFrom method</td>
<td>Copies the properties of the specified ULSyncParms object to this ULSyncParms object.</td>
</tr>
<tr>
<td>ToString method</td>
<td>Returns the string representation of this instance.</td>
</tr>
<tr>
<td>AdditionalParms property</td>
<td>Specifies additional synchronization parameters as a semicolon-separated list of name=value pairs.</td>
</tr>
<tr>
<td>AuthenticationParms property</td>
<td>Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).</td>
</tr>
<tr>
<td>DownloadOnly property</td>
<td>Specifies whether to disable or enable uploads when synchronizing.</td>
</tr>
<tr>
<td>KeepPartialDownload property</td>
<td>Specifies whether to disable or enable partial downloads when synchronizing.</td>
</tr>
<tr>
<td>NewPassword property</td>
<td>Specifies a new MobiLink password for the user specified with Username.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Password property</td>
<td>The MobiLink password for the user specified by UserName.</td>
</tr>
<tr>
<td>PingOnly property</td>
<td>Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.</td>
</tr>
<tr>
<td>Publications property</td>
<td>Specifies the publications to be synchronized.</td>
</tr>
<tr>
<td>ResumePartialDownload prop-</td>
<td>Specifies whether to resume or discard a previous partial download.</td>
</tr>
<tr>
<td>ety</td>
<td></td>
</tr>
<tr>
<td>SendDownloadAck property</td>
<td>Specifies whether the client should send a download acknowledgement to the MobiLink server during synchronization.</td>
</tr>
<tr>
<td>Stream property</td>
<td>Specifies the MobiLink synchronization stream to use for synchronization.</td>
</tr>
<tr>
<td>StreamParms property</td>
<td>Specifies the parameters to configure the synchronization stream.</td>
</tr>
<tr>
<td>UploadOnly property</td>
<td>Specifies whether to disable or enable downloads when synchronizing.</td>
</tr>
<tr>
<td>UserName property</td>
<td>The user name that uniquely identifies the MobiLink client to the MobiLink server.</td>
</tr>
<tr>
<td>Version property</td>
<td>Specifies which synchronization script to use.</td>
</tr>
</tbody>
</table>

**Remarks**

There is no constructor for this class. Each connection has its own ULSyncParms instance, attached as its ULConnection.SyncParms property.

At most, only one synchronization command (the ULSyncParms.DownloadOnly, ULSyncParms.PingOnly, ULSyncParms.ResumePartialDownload, or ULSyncParms.UploadOnly property) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode.SQLE_SYNC_INFO_INVALID SQLException is thrown by the ULConnection.Synchronize method.

Other sources of ULSQLCode.SQLE_SYNC_INFO_INVALID errors include not specifying a ULSyncParms.Stream value or a ULSyncParms.Version value.
See also
- “ULConnection class [UltraLite.NET]” on page 118
- “ULConnection.SyncParms property [UltraLite.NET]” on page 160
- “ULConnection.Synchronize method [UltraLite.NET]” on page 150
- “ULSyncParms.DownloadOnly property [UltraLite.NET]” on page 379
- “ULSyncParms.PingOnly property [UltraLite.NET]” on page 381
- “ULSyncParms.ResumePartialDownload property [UltraLite.NET]” on page 382
- “ULSyncParms.UploadOnly property [UltraLite.NET]” on page 385
- “ULConnection.Synchronize method [UltraLite.NET]” on page 150
- “ULSyncParms.Stream property [UltraLite.NET]” on page 384
- “ULSyncParms.Version property [UltraLite.NET]” on page 386

**CopyFrom method**
Copies the properties of the specified ULSyncParms object to this ULSyncParms object.

**Visual Basic syntax**
```
Public Sub CopyFrom(ByVal src As ULSyncParms)
```

**C# syntax**
```
public void CopyFrom(ULSyncParms src)
```

**Parameters**
- `src` The object to copy from.

**See also**
- “ULSyncParms class [UltraLite.NET]” on page 375

**ToString method**
Returns the string representation of this instance.

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

**C# syntax**
```
public override string ToString()
```

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

**AdditionalParms property**
Specifies additional synchronization parameters as a semicolon-separated list of name=value pairs.
Visual Basic syntax

Public Property AdditionalParms As String

C# syntax

public string AdditionalParms {get;set;}

Returns

A string, in the form of a semicolon-separated list of name=value pairs.

Remarks

Use this property to specify several additional synchronization parameters that cannot be readily specified using any other predefined parameters.

See also

● “Additional Parameters synchronization parameter” (UltraLite - Database Management and Reference)

Example

private ULSyncParms info;
// ...
info.AdditionalParms =
"AllowDownloadDupRows=1;
CheckpointStore=1;
DisableConcurrency=1;
TableOrder=Customer,Sales"

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Visual Basic syntax

Public Property AuthenticationParms As String()

C# syntax

public string[] AuthenticationParms {get;set;}

Returns

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 the MobiLink server's limit for authentication parameters (currently 4000 UTF8 bytes).
DownloadOnly property

Specifies whether to disable or enable uploads when synchronizing.

Visual Basic syntax

Public Property DownloadOnly As Boolean

C# syntax

public bool DownloadOnly {get;set;}

Returns

True to disable uploads when synchronizing, false to enable uploads. The default is false.

Remarks

At most, only one synchronization command (the ULSyncParms.DownloadOnly, ULSyncParms.PingOnly, ULSyncParms.ResumePartialDownload, or ULSyncParms.UploadOnly property) can be specified at a time. If more than one of these parameters is set to true, a ULSQLeCode.SQLE_SYNC_INFO_INVALID SQLException is thrown by the ULConnection.Synchronize method.

See also

- “ULSyncParms.UploadOnly property [UltraLite.NET]” on page 385
- “ULSyncParms.DownloadOnly property [UltraLite.NET]” on page 379
- “ULSyncParms.PingOnly property [UltraLite.NET]” on page 381
- “ULSyncParms.ResumePartialDownload property [UltraLite.NET]” on page 382
- “ULSyncParms.UploadOnly property [UltraLite.NET]” on page 385
- “ULConnection.Synchronize method [UltraLite.NET]” on page 150

KeepPartialDownload property

Specifies whether to disable or enable partial downloads when synchronizing.

Visual Basic syntax

Public Property KeepPartialDownload As Boolean

C# syntax

public bool KeepPartialDownload {get;set;}

Returns

Set to true to enable and save partial downloads while synchronizing; otherwise, set to false to disable partial downloads and roll back downloads if any errors occur. The default is false.

Remarks

Using the ULSyncProgressListener object, UltraLite.NET can resume partial downloads that fail because of communication errors or user aborts. 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.

If a partial download was kept, then the ULConnection.ULSyncResult.PartialDownloadRetained property is set to true when the ULConnection.Synchronize method exits.

If the PartialDownloadRetained property is set, then you can resume a download. To do this, call the ULConnection.Synchronize method with the ULConnection.ULSyncParms.ResumePartialDownload property set to true. It is recommended that you keep the KeepPartialDownload property set to true 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 recent data, then you can do another download immediately after the resumed download completes.

When resuming a download, many of the ULSyncParms properties are not relevant. For example, the Publications property is not used. You receive the publications that you requested on the initial download. The only properties that must be set are ResumePartialDownload and UserName. The KeepPartialDownload property can be set if desired and functions as normal.

If you have a partial download that is no longer needed, call the ULConnection.RollbackPartialDownload method to roll back the failed download transaction. If you attempt to synchronize again and do not specify the ResumePartialDownload property, then the partial download is rolled back before the next synchronization begins.

See also
- “ULSyncResult.PartialDownloadRetained property [UltraLite.NET]” on page 399
- “ULSyncParms.ResumePartialDownload property [UltraLite.NET]” on page 382
- “ULConnection.RollbackPartialDownload method [UltraLite.NET]” on page 148
- “ULSyncParms.ResumePartialDownload property [UltraLite.NET]” on page 382
- “ULSyncParms.UserName property [UltraLite.NET]” on page 385
- “ULConnection.RollbackPartialDownload method [UltraLite.NET]” on page 148
- “Resumption of failed downloads” [MobiLink - Server Administration]

**NewPassword property**

Specifies a new MobiLink password for the user specified with UserName.

**Visual Basic syntax**

```vbnet
Public Property NewPassword As String
```

**C# syntax**

```csharp
public string NewPassword {get;set;}
```

**Returns**

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.UserName property [UltraLite.NET]” on page 385

Password property
The MobiLink password for the user specified by UserName.

Visual Basic syntax
Public Property Password As String

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

Returns
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.NewPassword property [UltraLite.NET]” on page 380
● “ULSyncParms.UserName property [UltraLite.NET]” on page 385

PingOnly property
Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.

Visual Basic syntax
Public Property PingOnly As Boolean

C# syntax
public bool PingOnly {get;set;}

Returns
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 (the ULSyncParms.DownloadOnly, ULSyncParms.PingOnly, ULSyncParms.ResumePartialDownload, or ULSyncParms.UploadOnly property) can be specified at a time. If more than one of these parameters is set to true, a ULSQLException.SQLE_SYNC_INFO_INVALID SQLException is thrown by the ULConnection.Synchronize method.

See also

- “ULSyncParms.DownloadOnly property [UltraLite.NET]” on page 379
- “ULSyncParms.PingOnly property [UltraLite.NET]” on page 381
- “ULSyncParms.ResumePartialDownload property [UltraLite.NET]” on page 382
- “ULSyncParms.UploadOnly property [UltraLite.NET]” on page 385
- “ULConnection.Synchronize method [UltraLite.NET]” on page 150

Publications property

Specifies the publications to be synchronized.

Visual Basic syntax

Public Property Publications As String

C# syntax

public string Publications {get;set;}

Returns

A string containing a list of publication names, separated by comma (,); or the special value ULConnection.SYNC_ALL_PUBS, or the special value ULConnection.SYNC_ALL_DB. The default is ULConnection.SYNC_ALL_DB.

See also

- “ULConnection.SYNC_ALL_PUBS field [UltraLite.NET]” on page 163
- “ULConnection.SYNC_ALL_DB field [UltraLite.NET]” on page 163

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Visual Basic syntax

Public Property ResumePartialDownload As Boolean

C# syntax

public bool ResumePartialDownload {get;set;}

Returns

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 (the ULSyncParms.DownloadOnly, ULSyncParms.PingOnly, ULSyncParms.ResumePartialDownload, or ULSyncParms.UploadOnly property) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode.SQLE_SYNC_INFO_INVALID SQLException is thrown by the ULConnection.Synchronize method.

See also

- “ULSyncParms.KeepPartialDownload property [UltraLite.NET]” on page 379
- “ULSyncParms.DownloadOnly property [UltraLite.NET]” on page 379
- “ULSyncParms.PingOnly property [UltraLite.NET]” on page 381
- “ULSyncParms.ResumePartialDownload property [UltraLite.NET]” on page 382
- “ULSyncParms.UploadOnly property [UltraLite.NET]” on page 385
- “ULConnection.Synchronize method [UltraLite.NET]” on page 150
- “ULSyncResult.PartialDownloadRetained property [UltraLite.NET]” on page 399

SendDownloadAck property

Specifies whether the client should send a download acknowledgement to the MobiLink server during synchronization.

Visual Basic syntax

Public Property SendDownloadAck As Boolean

C# syntax

public bool SendDownloadAck {get;set;}

Returns

Set to true to specify that the client should send a download acknowledgement to the MobiLink server. Set to false to specify that no download acknowledgement is sent. The default is false.

Remarks

The download acknowledgement is sent after the download has been fully applied and committed at the remote (a positive acknowledgement) or after the download fails (a negative acknowledgement).

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 send a download acknowledgement, the MobiLink server is freed up sooner for its next synchronization.
Stream property

Specifies the MobiLink synchronization stream to use for synchronization.

Visual Basic syntax

Public Property Stream As ULStreamType

C# syntax

public ULStreamType Stream {get;set;}

Returns

One of the ULStreamType values specifying the type of synchronization stream to use. The default value is ULStreamType.TCPIP.

Remarks

Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. These parameters are supplied by the ULSyncParms.StreamParms property.

If the stream type is set to a value that is invalid for the platform, the stream type is set to ULStreamType.TCPIP.

See also

● “ULStreamType enumeration [UltraLite.NET]” on page 448
● “ULSyncParms.StreamParms property [UltraLite.NET]” on page 384

StreamParms property

Specifies the parameters to configure the synchronization stream.

Visual Basic syntax

Public Property StreamParms As String

C# syntax

public string StreamParms {get;set;}

Returns

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

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.Stream property [UltraLite.NET]" on page 384
- "ULStreamType enumeration [UltraLite.NET]" on page 448
- "UltraLite network protocol options" [UltraLite - Database Management and Reference]

UploadOnly property

Specifies whether to disable or enable downloads when synchronizing.

Visual Basic syntax

Public Property UploadOnly As Boolean

C# syntax

public bool UploadOnly {get;set;}

Returns

True to disable downloads, false to enable downloads. The default is false.

Remarks

At most, only one synchronization command (the ULSyncParms.DownloadOnly, ULSyncParms.PingOnly, ULSyncParms.ResumePartialDownload, or ULSyncParms.UploadOnly property) can be specified at a time. If more than one of these parameters is set to true, a ULSQLCode.SQLE_SYNC_INFO_INVALID SQLException is thrown by the ULConnection.Synchronize method.

See also

- "ULSyncParms.DownloadOnly property [UltraLite.NET]" on page 379
- "ULSyncParms.PingOnly property [UltraLite.NET]" on page 381
- "ULSyncParms.ResumePartialDownload property [UltraLite.NET]" on page 382
- "ULSyncParms.UploadPartialDownload property [UltraLite.NET]" on page 385
- "ULConnection.Synchronize method [UltraLite.NET]" on page 150

UserName property

The user name that uniquely identifies the MobiLink client to the MobiLink server.

Visual Basic syntax

Public Property UserName As String

C# syntax

public string UserName {get;set;}

Returns

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.Password property [UltraLite.NET]” on page 381

Version property
Specifies which synchronization script to use.

Visual Basic syntax
Public Property Version As String

C# syntax
public string Version {get;set;}

Returns
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.

ULSyncProgressData class
UL Ext: Returns synchronization progress monitoring data.

Visual Basic syntax
Public Class ULSyncProgressData

C# syntax
public class ULSyncProgressData

Members
All members of the ULSyncProgressData class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentDownloadRowCount property</td>
<td>Returns the number of rows that have been downloaded so far.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Flags property</td>
<td>Returns the current synchronization flags indicating additional information relating to the current state.</td>
</tr>
<tr>
<td>IgnoredDeletes property</td>
<td>Returns the number of rows received so far that have already been deleted.</td>
</tr>
<tr>
<td>IgnoredUpdates property</td>
<td>Returns the number of rows received so far that have already been updated.</td>
</tr>
<tr>
<td>IsFinalSyncProgress property</td>
<td>Returns true if this is final synchronization progress message.</td>
</tr>
<tr>
<td>ReceivedBytes property</td>
<td>Returns the number of bytes received so far.</td>
</tr>
<tr>
<td>ReceivedDeletes property</td>
<td>Returns the number of deleted rows received so far.</td>
</tr>
<tr>
<td>ReceivedInserts property</td>
<td>Returns the number of inserted rows received so far.</td>
</tr>
<tr>
<td>ReceivedUpdates property</td>
<td>Returns the number of updated rows received so far.</td>
</tr>
<tr>
<td>SentBytes property</td>
<td>Returns the number of bytes sent so far.</td>
</tr>
<tr>
<td>SentDeletes property</td>
<td>Returns the number of deleted rows sent so far.</td>
</tr>
<tr>
<td>SentInserts property</td>
<td>Returns the number of inserted rows sent so far.</td>
</tr>
<tr>
<td>SentUpdates property</td>
<td>Returns the number of updated rows sent so far.</td>
</tr>
<tr>
<td>State property</td>
<td>Returns the current synchronization state.</td>
</tr>
<tr>
<td>SyncTableCount property</td>
<td>Returns the number of tables being synchronized.</td>
</tr>
<tr>
<td>SyncTableIndex property</td>
<td>Returns the index of the table currently being synchronized in the range from 1 to the total number of tables involved with the synchronization.</td>
</tr>
<tr>
<td>TableID property</td>
<td>Returns the database index of the table currently being synchronized.</td>
</tr>
<tr>
<td>TableName property</td>
<td>Returns the name of the current table being uploaded or downloaded.</td>
</tr>
<tr>
<td>TotalDownloadRowCount property</td>
<td>Returns the total number of rows to be received in the download.</td>
</tr>
<tr>
<td>TruncateDeletes property</td>
<td>Returns the number of rows that have been deleted by a truncate operation.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FLAG_IS_BLOCKING field</td>
<td>A flag indicating that the synchronization is blocked awaiting a response from the MobiLink server.</td>
</tr>
</tbody>
</table>

**See also**

- “ULSyncProgressListener interface [UltraLite.NET]” on page 396

## CurrentDownloadRowCount property

Returns the number of rows that have been downloaded so far.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property CurrentDownloadRowCount As Integer
```

**C# syntax**

```csharp
public int CurrentDownloadRowCount {get;}
```

**Returns**

The number of rows that have been downloaded so far.

**Remarks**

This number includes duplicate rows that aren't included in ReceivedInserts, ReceivedUpdates, or ReceivedDeletes.

**See also**

- “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

## Flags property

Returns the current synchronization flags indicating additional information relating to the current state.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property Flags As Integer
```

**C# syntax**

```csharp
public int Flags {get;}
```

**Returns**

An integer containing a combination of flags or'ed together.

**See also**

- “ULSyncProgressData.FLAG_IS_BLOCKING field [UltraLite.NET]” on page 395
**IgnoredDeletes property**

Returns the number of rows received so far that have already been deleted.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property IgnoredDeletes As Integer
```

**C# syntax**

```csharp
public int IgnoredDeletes {get;}
```

**Returns**

The number of rows received so far that have already been deleted.

**See also**

- “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

**IgnoredUpdates property**

Returns the number of rows received so far that have already been updated.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property IgnoredUpdates As Integer
```

**C# syntax**

```csharp
public int IgnoredUpdates {get;}
```

**Returns**

The number of rows received so far that have already been updated.

**See also**

- “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

**IsFinalSyncProgress property**

Returns true if this is final synchronization progress message.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property IsFinalSyncProgress As Boolean
```

**C# syntax**

```csharp
public bool IsFinalSyncProgress {get;}
```

**Returns**

True if this is the final synchronization progress message.
**ReceivedBytes property**

Returns the number of bytes received so far.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property ReceivedBytes As Long
```

**C# syntax**

```csharp
public long ReceivedBytes {get;}
```

**Returns**

The number of bytes received so far.

**Remarks**

This information is updated for all states.

**See also**

- “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

---

**ReceivedDeletes property**

Returns the number of deleted rows received so far.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property ReceivedDeletes As Integer
```

**C# syntax**

```csharp
public int ReceivedDeletes {get;}
```

**Returns**

The number of deleted rows received so far.

**See also**

- “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

---

**ReceivedInserts property**

Returns the number of inserted rows received so far.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property ReceivedInserts As Integer
```

**C# syntax**

```csharp
public int ReceivedInserts {get;}
```
Returns

The number of inserted rows received so far.

See also

- “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

### ReceivedUpdates property

Returns the number of updated rows received so far.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property ReceivedUpdates As Integer
```

**C# syntax**

```csharp
public int ReceivedUpdates {get;}
```

Returns

The number of updated rows received so far.

See also

- “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

### SentBytes property

Returns the number of bytes sent so far.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property SentBytes As Long
```

**C# syntax**

```csharp
public long SentBytes {get;}
```

Returns

The number of bytes sent so far.

**Remarks**

This information is updated for all states.

See also

- “ULSyncProgressState enumeration [UltraLite.NET]” on page 449
SentDeletes property
Returns the number of deleted rows sent so far.

Visual Basic syntax
Public ReadOnly Property SentDeletes As Integer

C# syntax
public int SentDeletes {get;}

Returns
The number of deleted rows sent so far.

See also
● “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

SentInserts property
Returns the number of inserted rows sent so far.

Visual Basic syntax
Public ReadOnly Property SentInserts As Integer

C# syntax
public int SentInserts {get;}

Returns
The number of inserted rows sent so far.

See also
● “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

SentUpdates property
Returns the number of updated rows sent so far.

Visual Basic syntax
Public ReadOnly Property SentUpdates As Integer

C# syntax
public int SentUpdates {get;}

Returns
The number of updated rows sent so far.
State property

Returns the current synchronization state.

Visual Basic syntax

Public ReadOnly Property State As ULSyncProgressState

C# syntax

public ULSyncProgressState State {get;}

Returns

One of the ULSyncProgressState values specifying the current synchronization state.

See also

● “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

SyncTableCount property

Returns the number of tables being synchronized.

Visual Basic syntax

Public ReadOnly Property SyncTableCount As Integer

C# syntax

public int SyncTableCount {get;}

Returns

The number of tables being synchronized. For each table there is a sending and receiving phase, so this number may be more than the number of tables being synchronized.

See also

● “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

SyncTableIndex property

Returns the index of the table currently being synchronized in the range from 1 to the total number of tables involved with the synchronization.

Visual Basic syntax

Public ReadOnly Property SyncTableIndex As Integer
C# syntax

```csharp
public int SyncTableIndex {get;}
```

Returns

The index of the table currently being synchronized in the range from 1 to the SyncTableCount property value.

See also

● “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

### TableID property

Returns the database index of the table currently being synchronized.

Visual Basic syntax

```vbnet
Public ReadOnly Property TableID As Integer
```

C# syntax

```csharp
public int TableID {get;}
```

Returns

The database index, in the range from 1 to the ULDatabaseSchema.TableCount property value.

See also

● “ULSyncProgressState enumeration [UltraLite.NET]” on page 449  
● “ULDatabaseSchema.TableCount property [UltraLite.NET]” on page 228

### TableName property

Returns the name of the current table being uploaded or downloaded.

Visual Basic syntax

```vbnet
Public ReadOnly Property TableName As String
```

C# syntax

```csharp
public string TableName {get;}
```

Returns

Name of the current table being synchronized; null if not applicable.

### TotalDownloadRowCount property

Returns the total number of rows to be received in the download.
Visual Basic syntax
Public ReadOnly Property TotalDownloadRowCount As Integer

C# syntax
public int TotalDownloadRowCount {get;}

Returns
The number of rows to be received in the download.

Remarks
This number includes duplicate rows that aren't included in ReceivedInserts, ReceivedUpdates, or ReceivedDeletes. This value isn't set until the synchronization enters the STATE_RECEIVING_TABLE state for the first table.

See also
● “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

TruncateDeletes property
Returns the number of rows that have been deleted by a truncate operation.

Visual Basic syntax
Public ReadOnly Property TruncateDeletes As Integer

C# syntax
public int TruncateDeletes {get;}

Returns
The number of rows deleted by a truncate operation.

See also
● “ULSyncProgressState enumeration [UltraLite.NET]” on page 449

FLAG_IS_BLOCKING field
A flag indicating that the synchronization is blocked awaiting a response from the MobiLink server.

Visual Basic syntax
Public Const FLAG_IS_BLOCKING As Integer

C# syntax
public const int FLAG_IS_BLOCKING;
ULSyncProgressListener interface

**UL Ext:** The listener interface for receiving synchronization progress events.

**Visual Basic syntax**

```vbnet
Public Interface ULSyncProgressListener

**C# syntax**

```csharp
public interface ULSyncProgressListener

**Members**

All members of the ULSyncProgressListener interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SyncProgressed method</td>
<td>Invoked during synchronization to inform the user of progress.</td>
</tr>
</tbody>
</table>

**See also**

- “ULConnection.Synchronize method [UltraLite.NET]” on page 150

SyncProgressed method

Invoked during synchronization to inform the user of progress.

**Visual Basic syntax**

```vbnet
Public Function SyncProgressed(
    ByVal data As ULSyncProgressData
) As Boolean

**C# syntax**

```csharp
public bool SyncProgressed(ULSyncProgressData data)

**Parameters**

- **data** A ULSyncProgressData object containing the latest synchronization progress data.

**Returns**

This method should return true to cancel synchronization or return false to continue.

**Remarks**

This method should return true to cancel synchronization or return false to continue.

No UltraLite.NET API methods should be invoked during a SyncProgressed call.

**See also**

- “ULSyncProgressData class [UltraLite.NET]” on page 386
ULSyncResult class

UL Ext: Represents the status of the last synchronization.

Visual Basic syntax

Public Class ULSyncResult

C# syntax

public class ULSyncResult

Members

All members of the ULSyncResult class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthStatus property</td>
<td>Returns the authorization status code for the last synchronization attempt.</td>
</tr>
<tr>
<td>AuthValue property</td>
<td>Returns the return value from custom user authentication synchronization scripts.</td>
</tr>
<tr>
<td>IgnoredRows property</td>
<td>Checks whether any uploaded rows were ignored during the last synchronization.</td>
</tr>
<tr>
<td>PartialDownloadRetained prop-</td>
<td>Checks whether a partial download was retained during the last synchronization.</td>
</tr>
<tr>
<td>erty</td>
<td></td>
</tr>
<tr>
<td>StreamErrorCode property</td>
<td>Returns the error reported by the stream itself.</td>
</tr>
<tr>
<td>StreamErrorParameters prop-</td>
<td>Returns a comma-separated list of stream error parameters.</td>
</tr>
<tr>
<td>erty</td>
<td></td>
</tr>
<tr>
<td>StreamErrorSystem property</td>
<td>Returns the stream error system-specific code.</td>
</tr>
<tr>
<td>Timestamp property</td>
<td>Returns the timestamp of the last synchronization.</td>
</tr>
<tr>
<td>UploadOK property</td>
<td>Checks whether the last upload synchronization was successful.</td>
</tr>
</tbody>
</table>

Remarks

There is no constructor for this class. Each connection has its own ULSyncResult instance, attached as its ULConnection.SyncResult property. A ULSyncResult instance is only valid while that connection is open.

See also

- “ULConnection.SyncResult property [UltraLite.NET]” on page 160
- “ULConnection.Synchronize method [UltraLite.NET]” on page 150
**AuthStatus property**

Returns the authorization status code for the last synchronization attempt.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property AuthStatus As ULAuthStatusCode
```

**C# syntax**

```csharp
public ULAuthStatusCode AuthStatus {get;}
```

**Remarks**

One of the ULAuthStatusCode values denoting the authorization status for the last synchronization attempt.

**See also**

- “ULAuthStatusCode enumeration [UltraLite.NET]” on page 440

**AuthValue property**

Returns the return value from custom user authentication synchronization scripts.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property AuthValue As Long
```

**C# syntax**

```csharp
public long AuthValue {get;}
```

**Remarks**

A long integer returned from custom user authentication synchronization scripts.

**IgnoredRows property**

Checks whether any uploaded rows were ignored during the last synchronization.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property IgnoredRows As Boolean
```

**C# syntax**

```csharp
public bool IgnoredRows {get;}
```

**Remarks**

True if any uploaded rows were ignored during the last synchronization, false if no rows were ignored.
See also

- “ULSyncParms.DownloadOnly property [UltraLite.NET]” on page 379

PartialDownloadRetained property
Checks whether a partial download was retained during the last synchronization.

Visual Basic syntax
Public ReadOnly Property PartialDownloadRetained As Boolean

C# syntax
public bool PartialDownloadRetained {get;}

Remarks
True if a download was interrupted and the partial download was retained, false if the download was not interrupted or if the partial download was rolled back.

See also

- “ULSyncParms.KeepPartialDownload property [UltraLite.NET]” on page 379

StreamErrorCode property
Returns the error reported by the stream itself.

Visual Basic syntax
Public ReadOnly Property StreamErrorCode As ULStreamErrorCode

C# syntax
public ULStreamErrorCode StreamErrorCode {get;}

Remarks
One of the ULStreamErrorCode values denoting the error reported by the stream itself, ULStreamErrorCode.NONE if no error occurred.

StreamErrorParameters property
Returns a comma-separated list of stream error parameters.

Visual Basic syntax
Public ReadOnly Property StreamErrorParameters As String

C# syntax
public string StreamErrorParameters {get;}
Remarks
Contains a comma separated list of error parameters for the stream error code reported in StreamErrorCode property. This is an empty string either for errors with no parameters, or when no error has been set.

See also
- “ULFileTransfer.StreamErrorCode property [UltraLite.NET]” on page 284

StreamErrorSystem property
Returns the stream error system-specific code.

Visual Basic syntax
Public ReadOnly Property StreamErrorSystem As Integer

C# syntax
public int StreamErrorSystem {get;}

Remarks
An integer denoting the stream error system-specific code.

Timestamp property
Returns the timestamp of the last synchronization.

Visual Basic syntax
Public ReadOnly Property Timestamp As Date

C# syntax
public DateTime Timestamp {get;}

Remarks
A System.DateTime structure specifying the timestamp of the last synchronization.

See also
- System.DateTime

UploadOK property
Checks whether the last upload synchronization was successful.

Visual Basic syntax
Public ReadOnly Property UploadOK As Boolean
C# syntax

    public bool UploadOK {get;}

Remarks

    True if the last upload synchronization was successful, false if the last upload synchronization was unsuccessful.

ULTable class

UL Ext: Represents a table in an UltraLite database.

Visual Basic syntax

    Public Class ULTable Inherits ULResultSet

C# syntax

    public class ULTable : ULResultSet

Base classes

    ● “ULResultSet class [UltraLite.NET]” on page 339

Members

    All members of the ULTable class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppendBytes method</td>
<td>Appends the specified subset of the specified array of System.Bytes to the new value for the specified ULDDbType.LongBinary column.</td>
</tr>
<tr>
<td>AppendChars method</td>
<td>Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDDbType.LongVarchar column.</td>
</tr>
<tr>
<td>Close method</td>
<td>Closes the cursor.</td>
</tr>
<tr>
<td>Delete method</td>
<td>Deletes the current row.</td>
</tr>
<tr>
<td>DeleteAllRows method</td>
<td>Deletes all rows in the table.</td>
</tr>
<tr>
<td>FindBegin method</td>
<td>Prepares to perform a new Find on a table.</td>
</tr>
<tr>
<td>FindFirst method</td>
<td>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.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FindLast method</td>
<td>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.</td>
</tr>
<tr>
<td>FindNext method</td>
<td>Continues a ULTable.FindFirst search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.</td>
</tr>
<tr>
<td>FindPrevious method</td>
<td>Continues a ULTable.FindLast search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.</td>
</tr>
<tr>
<td>GetBoolean method</td>
<td>Returns the value for the specified column as a System.Boolean.</td>
</tr>
<tr>
<td>GetByte method</td>
<td>Returns the value for the specified column as an unsigned 8-bit value (System.Byte).</td>
</tr>
<tr>
<td>GetBytes method</td>
<td><strong>UL Ext:</strong> Returns the value for the specified column as an array of System.Byte values.</td>
</tr>
<tr>
<td>GetChar method</td>
<td>This method is not supported in UltraLite.NET.</td>
</tr>
<tr>
<td>GetChars method</td>
<td>Copies a subset of the value for the specified ULDbType.LongVarchar column, beginning at the specified offset, to the specified offset of the destination System.Char array.</td>
</tr>
<tr>
<td>GetDataTypeName method</td>
<td>Returns the name of the specified column's provider data type.</td>
</tr>
<tr>
<td>GetDateTime method</td>
<td>Returns the value for the specified column as a System.DateTime type with millisecond accuracy.</td>
</tr>
<tr>
<td>GetDecimal method</td>
<td>Returns the value for the specified column as a System.Decimal type.</td>
</tr>
<tr>
<td>GetDouble method</td>
<td>Returns the value for the specified column as a System.Double type.</td>
</tr>
<tr>
<td>GetEnumerator method</td>
<td>Returns an System.Collections.IEnumerator value that iterates through the ULDDataReader object.</td>
</tr>
<tr>
<td>GetFieldType method</td>
<td>Returns the System.Type value most appropriate for the specified column.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>GetFieldValue(T) method</td>
<td>Synchronously gets the value of the specified column as a type. (Inherited from System.Data.Common.DbDataReader)</td>
</tr>
<tr>
<td>GetFieldValueAsync method</td>
<td>Asynchronously gets the value of the specified column as a type. (Inherited from System.Data.Common.DbDataReader)</td>
</tr>
<tr>
<td>GetFloat method</td>
<td>Returns the value for the specified column as a System.Single type.</td>
</tr>
<tr>
<td>GetGuid method</td>
<td>Returns the value for the specified column as a UUID (System.Guid) type.</td>
</tr>
<tr>
<td>GetInt16 method</td>
<td>Returns the value for the specified column as a System.Int16 type.</td>
</tr>
<tr>
<td>GetInt32 method</td>
<td>Returns the value for the specified column as a System.Int32 type.</td>
</tr>
<tr>
<td>GetInt64 method</td>
<td>Returns the value for the specified column as a System.Int64 type.</td>
</tr>
<tr>
<td>GetName method</td>
<td>Returns the name of the specified column.</td>
</tr>
<tr>
<td>GetOrdinal method</td>
<td>Returns the column ID of the named column.</td>
</tr>
<tr>
<td>GetProviderSpecificFieldType method</td>
<td>Returns the provider-specific field type of the specified column. (Inherited from System.Data.Common.DbDataReader)</td>
</tr>
<tr>
<td>GetProviderSpecificValues method</td>
<td>Gets all provider-specific attribute columns in the collection for the current row. (Inherited from System.Data.Common.DbDataReader)</td>
</tr>
<tr>
<td>GetRowCount method</td>
<td>UL Ext: Returns the number of rows in the cursor, within threshold.</td>
</tr>
<tr>
<td>GetSchemaTable method</td>
<td>Returns a System.Data.DataTable value that describes the column metadata of the ULDataReader object.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetString method</td>
<td>Returns the value for the specified column as a System.String type.</td>
</tr>
<tr>
<td>GetTimeSpan method</td>
<td>Returns the value for the specified column as a System.TimeSpan type</td>
</tr>
<tr>
<td>GetUInt16 method</td>
<td>Returns the value for the specified column as a System.UInt16 type.</td>
</tr>
<tr>
<td>GetUInt32 method</td>
<td>Returns the value for the specified column as a System.UInt32 type.</td>
</tr>
<tr>
<td>GetUInt64 method</td>
<td>Returns the value for the specified column as a System.UInt64 type.</td>
</tr>
<tr>
<td>GetValue method</td>
<td>Returns the value of the specified column in its native format.</td>
</tr>
<tr>
<td>GetValues method</td>
<td>Returns all the column values for the current row.</td>
</tr>
<tr>
<td>Insert method</td>
<td>Inserts a new row with the current column values (specified using the set methods).</td>
</tr>
<tr>
<td>InsertBegin method</td>
<td>Prepares to insert a new row into the table by setting all current column values to their default values.</td>
</tr>
<tr>
<td>IsDBNull method</td>
<td>Checks whether the value from the specified column is NULL.</td>
</tr>
<tr>
<td>IsDBNullAsync method (Inherited from System.Data.Common.DbDataReader)</td>
<td>An asynchronous version of System.Data.Common.DbDataReader.IsDBNull(System.Int32), which gets a value that indicates whether the column contains non-existent or missing values.</td>
</tr>
<tr>
<td>LookupBackward method</td>
<td>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.</td>
</tr>
<tr>
<td>LookupBegin method</td>
<td>Prepares to perform a new lookup on the table.</td>
</tr>
<tr>
<td>LookupForward method</td>
<td>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.</td>
</tr>
<tr>
<td>MoveAfterLast method</td>
<td>UL Ext: Positions the cursor to after the last row of the cursor.</td>
</tr>
<tr>
<td>MoveBeforeFirst method</td>
<td>UL Ext: Positions the cursor to before the first row of the cursor.</td>
</tr>
<tr>
<td>MoveFirst method</td>
<td>UL Ext: Positions the cursor to the first row of the cursor.</td>
</tr>
<tr>
<td>MoveLast method</td>
<td>UL Ext: Positions the cursor to the last row of the cursor.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MoveNext method</td>
<td><strong>UL Ext:</strong> Positions the cursor to the next row or after the last row if the cursor was already on the last row.</td>
</tr>
<tr>
<td>MovePrevious method</td>
<td><strong>UL Ext:</strong> Positions the cursor to the previous row or before the first row.</td>
</tr>
<tr>
<td>MoveRelative method</td>
<td><strong>UL Ext:</strong> Positions the cursor relative to the current row.</td>
</tr>
<tr>
<td>NextResult method</td>
<td>Advances the ULDataReader object to the next result when reading the results of batch SQL statements.</td>
</tr>
<tr>
<td>Read method</td>
<td>Positions the cursor to the next row, or after the last row if the cursor was already on the last row.</td>
</tr>
<tr>
<td>ReadAsync method</td>
<td>An asynchronous version of System.Data.Common.DbDataReader.Read, which advances the reader to the next record in a result set.</td>
</tr>
<tr>
<td>SetBoolean method</td>
<td>Sets the value for the specified column using a System.Boolean.</td>
</tr>
<tr>
<td>SetByte method</td>
<td>Sets the value for the specified column using a System.Byte (unsigned 8-bit integer).</td>
</tr>
<tr>
<td>SetDateTime method</td>
<td>Sets the value for the specified column using a System.DateTime. Adamistenthe value for the specified column using a System.DateTime.</td>
</tr>
<tr>
<td>SetDBNull method</td>
<td>Sets a column to NULL.</td>
</tr>
<tr>
<td>SetDecimal method</td>
<td>Sets the value for the specified column using a System.Decimal. Adamistenthe value for the specified column using a System.Decimal.</td>
</tr>
<tr>
<td>SetDouble method</td>
<td>Sets the value for the specified column using a System.Double. Adamistenthe value for the specified column using a System.Double.</td>
</tr>
<tr>
<td>SetGuid method</td>
<td>Sets the value for the specified column using a System.Guid. Adamistenthe value for the specified column using a System.Guid.</td>
</tr>
<tr>
<td>SetInt32 method</td>
<td>Sets the value for the specified column using a System.Int32. Adamistenthe value for the specified column using a System.Int32.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SetInt64 method</td>
<td>Sets the value for the specified column using an Int64.</td>
</tr>
<tr>
<td>SetString method</td>
<td>Sets the value for the specified column using a System.String.</td>
</tr>
<tr>
<td>SetTimeSpan method</td>
<td>Sets the value for the specified column using a System.TimeSpan.</td>
</tr>
<tr>
<td>SetToDefault method</td>
<td>Sets the value for the specified column to its default value.</td>
</tr>
<tr>
<td>SetUInt16 method</td>
<td>Sets the value for the specified column using a System.UInt16.</td>
</tr>
<tr>
<td>SetUInt32 method</td>
<td>Sets the value for the specified column using a System.UInt32.</td>
</tr>
<tr>
<td>SetUInt64 method</td>
<td>Sets the value for the specified column using a System.UInt64.</td>
</tr>
<tr>
<td>Truncate method</td>
<td>Deletes all rows in the table while temporarily activating a stop synchroni-</td>
</tr>
<tr>
<td></td>
<td>zation delete.</td>
</tr>
<tr>
<td>Update method</td>
<td>Updates the current row with the current column values (specified using the</td>
</tr>
<tr>
<td></td>
<td>set methods).</td>
</tr>
<tr>
<td>UpdateBegin method</td>
<td>Prepares to update the current row.</td>
</tr>
<tr>
<td>Depth property</td>
<td>Returns the depth of nesting for the current row.</td>
</tr>
<tr>
<td>FieldCount property</td>
<td>Returns the number of columns in the cursor.</td>
</tr>
<tr>
<td>HasRows property</td>
<td>Checks whether the ULDataReader object has one or more rows.</td>
</tr>
<tr>
<td>IsBOF property</td>
<td>UL Ext: Checks whether the current row position is before the first row.</td>
</tr>
<tr>
<td>IsClosed property</td>
<td>Checks whether the cursor is currently open.</td>
</tr>
<tr>
<td>IsEOF property</td>
<td>UL Ext: Checks whether the current row position is after the last row.</td>
</tr>
<tr>
<td>RecordsAffected property</td>
<td>Returns the number of rows changed, inserted, or deleted by execution of the</td>
</tr>
<tr>
<td></td>
<td>SQL statement.</td>
</tr>
<tr>
<td>RowCount property</td>
<td>UL Ext: Returns the number of rows in the cursor.</td>
</tr>
<tr>
<td>Schema property</td>
<td>Holds the table schema.</td>
</tr>
<tr>
<td>this property</td>
<td>Returns the value of the specified column in its native format.</td>
</tr>
<tr>
<td>VisibleFieldCount property</td>
<td>Gets the number of fields in the System.Data.Common.DbDataReader that are n</td>
</tr>
<tr>
<td>(Inherited from System.Data.</td>
<td>Data.Common.DbDataReader)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Remarks
There is no constructor for this class. Tables are created using the ULCommand.ExecuteTable method.

See also
● “ULCommand.ExecuteTable method [UltraLite.NET]” on page 98
● “ULCommand class [UltraLite.NET]” on page 71
● “ULResultSet class [UltraLite.NET]” on page 339
● System.Data.IDataReader
● System.Data.IDataRecord
● System.IDisposable

DeleteAllRows method
Deletes all rows in the table.

Visual Basic syntax
Public Sub DeleteAllRows()

C# syntax
public void DeleteAllRows()

Exceptions
● ULException class  A SQL error occurred.

Remarks
In some applications, it can be useful to delete all rows from a table before downloading a new set of data into the table. Rows can be deleted from the UltraLite database without being deleted from the consolidated database using the ULConnection.StopSynchronizationDelete method.

See also
● “ULTable.Truncate method [UltraLite.NET]” on page 420
● “ULConnection.StopSynchronizationDelete method [UltraLite.NET]” on page 150

FindBegin method
Prepares to perform a new Find on a table.

Visual Basic syntax
Public Sub FindBegin()

C# syntax
public void FindBegin()
Exceptions

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

Remarks

The value(s) for which to search are specified by calling the appropriate setType method(s) on the columns in the index with which the table was opened.

See also

- “ULTable.FindFirst method [UltraLite.NET]” on page 408
- “ULTable.FindLast method [UltraLite.NET]” on page 410

**FindFirst method**

Moves forward through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindFirst() method</td>
<td>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.</td>
</tr>
<tr>
<td>FindFirst(short) method</td>
<td>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.</td>
</tr>
</tbody>
</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.

**Visual Basic syntax**

```vbnet
Public Function FindFirst() As Boolean
```

**C# syntax**

```csharp
public bool FindFirst()
```

**Returns**

True if successful, false otherwise.

**Exceptions**

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

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure, the cursor position is after the last row (as shown by the ULDataReader.IsEOF property).

The FindBegin method must be called before each search.

See also

- “ULTable.FindBegin method [UltraLite.NET]” on page 407
- “ULTable.FindNext method [UltraLite.NET]” on page 412
- “ULTable.FindPrevious method [UltraLite.NET]” on page 414
- “ULTable.FindFirst method [UltraLite.NET]” on page 408
- “ULDataReader.IsEOF property [UltraLite.NET]” on page 261
- “ULTable.FindBegin method [UltraLite.NET]” on page 407

FindFirst(short) method

Moves forward through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.

Visual Basic syntax

Public Function FindFirst(ByVal numColumns As Short) As Boolean

C# syntax

public bool FindFirst(short numColumns)

Parameters

- **numColumns** For composite indexes, the number of columns to use in the find. For example, if you have a three column index and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a value of 1.

Returns

True if successful, false otherwise.

Exceptions

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

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure, the cursor position is after the last row (as shown by the ULDataReader.IsEOF property).

The FindBegin method must be called before each search.
FindLast method

Moves backward through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindLast() method</td>
<td>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.</td>
</tr>
<tr>
<td>FindLast(short) method</td>
<td>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.</td>
</tr>
</tbody>
</table>

FindLast() method

Moves backward through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.

Visual Basic syntax

```vbnet
Public Function FindLast() As Boolean
```

C# syntax

```csharp
public bool FindLast()
```

Returns

True if successful, false otherwise.

Exceptions

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

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is before the first row (as shown by the ULDataReader.IsBOF property).
The FindBegin method must be called before each search.

See also
- “ULTable.FindBegin method [UltraLite.NET]” on page 407
- “ULTable.FindNext method [UltraLite.NET]” on page 412
- “ULTable.FindPrevious method [UltraLite.NET]” on page 414
- “ULTable.FindLast method [UltraLite.NET]” on page 410
- “ULDataReader.IsBOF property [UltraLite.NET]” on page 261
- “ULTable.FindBegin method [UltraLite.NET]” on page 407

FindLast(short) method
Moves backward through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.

Visual Basic syntax
Public Function FindLast(ByVal numColumns As Short) As Boolean

C# syntax
public bool FindLast(short numColumns)

Parameters
- numColumns For composite indexes, the number of columns to use in the find. For example, if you have a three column index and you want to find a value that matches based on the first column only, you should set the value for the first column, then supply a value of 1.

Returns
True if successful, false otherwise

Exceptions
- ULException class A SQL error occurred.

Remarks
To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is before the first row (as shown by the ULDataReader.IsBOF property).

The FindBegin method must be called before each search.
See also
- “ULTable.FindBegin method [UltraLite.NET]” on page 407
- “ULTable.FindNext method [UltraLite.NET]” on page 412
- “ULTable.FindPrevious method [UltraLite.NET]” on page 414
- “ULTable.FindLast method [UltraLite.NET]” on page 410
- “ULDataReader.IsBOF property [UltraLite.NET]” on page 261
- “ULTable.FindBegin method [UltraLite.NET]” on page 407

FindNext method
Continues a ULTable.FindFirst search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindNext() method</td>
<td>Continues a ULTable.FindFirst search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.</td>
</tr>
<tr>
<td>FindNext(short) method</td>
<td>Continues a ULTable.FindFirst search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or partial set of values in the current index.</td>
</tr>
</tbody>
</table>

FindNext() method
Continues a ULTable.FindFirst search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.

Visual Basic syntax
Public Function FindNext() As Boolean

C# syntax
public bool FindNext();

Returns
True if successful, false otherwise.

Exceptions
- ULException class A SQL error occurred.

Remarks
The cursor is left on the next row if it exactly matches the index value. On failure, the cursor position is after the last row (as shown by the ULDataReader.IsEOF property).
FindNext method behavior is undefined if the column values being searched for are modified during a row update.

**See also**
- “ULTable.FindFirst method [UltraLite.NET]” on page 408
- “ULTable.FindNext method [UltraLite.NET]” on page 412
- “ULDataReader.IsEOF property [UltraLite.NET]” on page 261

**FindNext(short) method**
Continues a ULTable.FindFirst search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or partial set of values in the current index.

**Visual Basic syntax**
```vbnet
Public Function FindNext(ByVal numColumns As Short) As Boolean
```

**C# syntax**
```csharp
public bool FindNext(short numColumns)
```

**Parameters**
- **numColumns** For composite indexes, the number of columns to use in the find. For example, if you have a three column index, and you want to find a value that matches based on the first column only, you should set the value for the first column, and then supply a value of 1.

**Returns**
True if successful, false otherwise.

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

**Remarks**
The cursor is left on the next row if it exactly matches the index value. On failure, the cursor position is after the last row (as shown by the ULDataReader.IsEOF property).

FindNext method behavior is undefined if the column values being searched for are modified during a row update.

**See also**
- “ULTable.FindFirst method [UltraLite.NET]” on page 408
- “ULTable.FindNext method [UltraLite.NET]” on page 412
- “ULTable.FindFirst method [UltraLite.NET]” on page 408
- “ULDataReader.IsEOF property [UltraLite.NET]” on page 261
**FindPrevious method**

Continues a ULTable.FindLast search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindPrevious() method</td>
<td>Continues a ULTable.FindLast search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.</td>
</tr>
<tr>
<td>FindPrevious(short) method</td>
<td>Continues a ULTable.FindLast search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or partial set of values in the current index.</td>
</tr>
</tbody>
</table>

**FindPrevious() method**

Continues a ULTable.FindLast search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.

**Visual Basic syntax**

```
Public Function FindPrevious() As Boolean
```

**C# syntax**

```
public bool FindPrevious()
```

**Returns**

True if successful, false otherwise.

**Exceptions**

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

**Remarks**

The cursor is left on the previous row if it exactly matches the index value. On failure, the cursor position is before the first row (as shown by the ULDataReader.IsBOF property).

FindPrevious method behavior is undefined if the column values being searched for are modified during a row update.

**See also**

- “ULTable.FindLast method [UltraLite.NET]” on page 410
- “ULTable.FindPrevious method [UltraLite.NET]” on page 414
- “ULDataReader.IsBOF property [UltraLite.NET]” on page 261
**FindPrevious(short) method**

Continues a ULTable.FindLast search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or partial set of values in the current index.

**Visual Basic syntax**

```vbnet
Public Function FindPrevious(ByVal numColumns As Short) As Boolean
```

**C# syntax**

```csharp
public bool FindPrevious(short numColumns)
```

**Parameters**

- **numColumns**  
  For composite indexes, the number of columns to use in the find. For example, if you have a three column index and you want to look up a value that matches based on the first column only, you should set the value for the first column, then supply a value of 1.

**Returns**

True if successful, false otherwise.

**Exceptions**

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

**Remarks**

The cursor is left on the previous row if it exactly matches the index value. On failure, the cursor position is before the first row (as shown by the ULDataReader.IsBOF property).

FindPrevious method behavior is undefined if the column values being searched for are modified during a row update.

**See also**

- “ULTable.FindLast method [UltraLite.NET]” on page 410
- “ULTable.FindPrevious method [UltraLite.NET]” on page 414
- “ULDataReader.IsBOF property [UltraLite.NET]” on page 261

**Insert method**

Inserts a new row with the current column values (specified using the set methods).

**Visual Basic syntax**

```vbnet
Public Sub Insert()
```

**C# syntax**

```csharp
public void Insert()
```
InsertBegin method

Prepares to insert a new row into the table by setting all current column values to their default values.

Visual Basic syntax

Public Sub InsertBegin()

C# syntax

public void InsertBegin()  

Exceptions

● **ULException class**  A SQL error occurred.

Remarks

Call the appropriate SetType or AppendType method(s) to specify the non-default values that are to be inserted.

The row is not actually inserted and the data in the row is not actually changed until you execute the Insert method, and that change is not made permanent until it is committed.

See also

● “ULTable.Insert method [UltraLite.NET]” on page 415

LookupBackward method

Moves backward through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LookupBackward()</td>
<td>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.</td>
</tr>
</tbody>
</table>
### LookupBackward() method

Moves backward through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.

**Visual Basic syntax**

```
Public Function LookupBackward() As Boolean
```

**C# syntax**

```
public bool LookupBackward()
```

**Returns**

True if successful, false otherwise.

**Exceptions**

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

**Remarks**

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that matches or is less than the index value. On failure (no rows less than the value being looked for), the cursor position is before the first row (as shown by the ULDataReader.IsBOF property).

The LookupBegin method must be called before each search.

**See also**

- “ULTable.LookupBegin method [UltraLite.NET]” on page 418
- “ULTable.LookupBackward method [UltraLite.NET]” on page 416
- “ULDataReader.IsBOF property [UltraLite.NET]” on page 261

### LookupBackward(short) method

Moves backward through the table from the end, looking for a row that matches or is less than a value or partial set of values in the current index.

**Visual Basic syntax**

```
Public Function LookupBackward(ByVal numColumns As Short) As Boolean
```

**C# syntax**

```
public bool LookupBackward(short numColumns)
```
Parameters

- **numColumns**  For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a value of 1.

Returns

True if successful, false otherwise.

Exceptions

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

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that matches or is less than the index value. On failure (no rows less than the value being looked for), the cursor position is before the first row (as shown by the ULDataReader.IsBOF property).

The LookupBegin method must be called before each search.

See also

- “ULTable.LookupForward method [UltraLite.NET]” on page 418
- “ULTable.LookupBackward method [UltraLite.NET]” on page 416
- “ULDataReader.IsBOF property [UltraLite.NET]” on page 261

**LookupBegin method**

Prepares to perform a new lookup on the table.

**Visual Basic syntax**

```vbnet
Public Sub LookupBegin()
```

**C# syntax**

```csharp
public void LookupBegin()
```

Exceptions

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

Remarks

The value(s) for which to search are specified by calling the appropriate setType method(s) on the columns in the index with which the table was opened.

See also

- “ULTable.LookupForward method [UltraLite.NET]” on page 419
- “ULTable.LookupBackward method [UltraLite.NET]” on page 416
**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LookupForward() method</td>
<td>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.</td>
</tr>
<tr>
<td>LookupForward(short) method</td>
<td>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.</td>
</tr>
</tbody>
</table>

**LookupForward() method**

Moves forward through the table from the beginning, looking for a row that matches or is greater than a value or full set of values in the current index.

**Visual Basic syntax**

```vbnet
Public Function LookupForward() As Boolean
```

**C# syntax**

```csharp
public bool LookupForward()
```

**Returns**

True if successful, false otherwise.

**Exceptions**

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

**Remarks**

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (no rows greater than the value being looked for), the cursor position is after the last row (as shown by the ULDataReader.IsEOF property).

The LookupBegin method must be called before each search.

**See also**

- “ULTable.LookupBegin method [UltraLite.NET]” on page 418
- “ULTable.LookupForward method [UltraLite.NET]” on page 419
- “ULDataReader.IsEOF property [UltraLite.NET]” on page 261
LookupForward(short) method

Moves forward through the table from the beginning, looking for a row that matches or is greater than a value or partial set of values in the current index.

Visual Basic syntax

```vbnet
Public Function LookupForward(ByVal numColumns As Short) As Boolean
```

C# syntax

```csharp
public bool LookupForward(short numColumns)
```

Parameters

- **numColumns**  For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a value of 1.

Returns

True if successful, false otherwise.

Exceptions

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

Remarks

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (no rows greater than the value being looked for), the cursor position is after the last row (as shown by the ULDataReader.IsEOF property).

The LookupBegin method must be called before each search.

See also

- “ULTable.LookupBegin method [UltraLite.NET]” on page 418
- “ULTable.LookupForward method [UltraLite.NET]” on page 419
- “ULDataReader.IsEOF property [UltraLite.NET]” on page 261
- “ULTable.LookupBegin method [UltraLite.NET]” on page 418

Truncate method

Deletes all rows in the table while temporarily activating a stop synchronization delete.

Visual Basic syntax

```vbnet
Public Sub Truncate()
```

C# syntax

```csharp
public void Truncate()
```
Exceptions

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

See also

- “ULTable.DeleteAllRows method [UltraLite.NET]” on page 407

**Schema property**

Holds the table schema.

**Visual Basic syntax**

```vbnet
Public ReadOnly Shadows Property Schema As ULTableSchema
```

**C# syntax**

```csharp
public new ULTableSchema Schema {get;}
```

**Remarks**

This property is only valid while its connection is open.

The ULTableSchema object representing the table schema.

This property represents the complete schema of the table, including UltraLite.NET extended information which is not represented in the results from calling the ULDataReader.GetSchemaTable method.

See also

- “ULTableSchema class [UltraLite.NET]” on page 421

**ULTableSchema class**

**UL Ext:** Represents the schema of an UltraLite table.

**Visual Basic syntax**

```vbnet
Public NotInheritable Class ULTableSchema Inherits ULCursorSchema
```

**C# syntax**

```csharp
public sealed class ULTableSchema : ULCursorSchema
```

**Base classes**

- “ULCursorSchema class [UltraLite.NET]” on page 197

**Members**

All members of the ULTableSchema class, including all inherited members.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetColumnDefaultValue method</td>
<td>Returns the default value of the specified column.</td>
</tr>
<tr>
<td>GetColumnID method</td>
<td>Returns the column ID of the named column.</td>
</tr>
<tr>
<td>GetColumnName method</td>
<td>Returns the name of the column identified by the specified column ID.</td>
</tr>
<tr>
<td>GetColumnPartitionSize method</td>
<td>Returns the global autoincrement partition size assigned to the specified column.</td>
</tr>
<tr>
<td>GetColumnPrecision method</td>
<td>Returns the precision of the column identified by the specified column ID if the column is a numeric column (the NUMERIC SQL type).</td>
</tr>
<tr>
<td>GetColumnScale method</td>
<td>Returns the scale of the column identified by the specified column ID if the column is a numeric column (the NUMERIC SQL type).</td>
</tr>
<tr>
<td>GetColumnSize method</td>
<td>Returns the size of the column identified by the specified column ID if the column is a sized column (the BINARY or CHAR SQL types).</td>
</tr>
<tr>
<td>GetColumnSQLName method</td>
<td>Returns the name of the column identified by the specified column ID.</td>
</tr>
<tr>
<td>GetColumnULDbType method</td>
<td>Returns the UltraLite.NET data type of the column identified by the specified column ID.</td>
</tr>
<tr>
<td>GetIndex method</td>
<td>Returns the index schema of the named index.</td>
</tr>
<tr>
<td>GetIndexName method</td>
<td>Returns the name of the index identified by the specified index ID.</td>
</tr>
<tr>
<td>GetOptimalIndex method</td>
<td>The optimal index for searching a table using the specified column.</td>
</tr>
<tr>
<td>GetPublicationPredicate method</td>
<td>Returns the publication predicate for this table in the named publication.</td>
</tr>
<tr>
<td>GetSchemaTable method</td>
<td>Returns a System.Data.DataTable that describes the column schema of the ULDataReader object.</td>
</tr>
<tr>
<td>IsColumnAutoIncrement method</td>
<td>Checks whether the specified column's default is set to autoincrement.</td>
</tr>
<tr>
<td>IsColumnCurrentDate method</td>
<td>Checks whether the specified column's default is set to the current date (a ULDbType.Date value).</td>
</tr>
<tr>
<td>IsColumnCurrentTime method</td>
<td>Checks whether the specified column's default is set to the current time (a ULDbType.Time value).</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsColumnCurrentTimestamp method</td>
<td>Checks whether the specified column's default is set to the current timestamp (a ULDbType.TimeStamp value).</td>
</tr>
<tr>
<td>IsColumnCurrentUTCTimestamp method</td>
<td>Checks whether the specified column’s default is set to the current UTC timestamp (a ULDbType.TimeStamp value).</td>
</tr>
<tr>
<td>IsColumnGlobalAutoIncrement method</td>
<td>Checks whether the specified column's default is set to global auto-increment.</td>
</tr>
<tr>
<td>IsColumnNewUUID method</td>
<td>Checks whether the specified column's default is set to a new UUID (a System.Guid value).</td>
</tr>
<tr>
<td>IsColumnNullable method</td>
<td>Checks whether the specified column is nullable.</td>
</tr>
<tr>
<td>IsInPublication method</td>
<td>Checks whether the table is contained in the named publication.</td>
</tr>
<tr>
<td>ColumnCount property</td>
<td>Returns the number of columns in the cursor.</td>
</tr>
<tr>
<td>IndexCount property</td>
<td>Returns the number of indexes on the table.</td>
</tr>
<tr>
<td>IsNeverSynchronized property</td>
<td>Checks whether the table is marked as never being synchronized.</td>
</tr>
<tr>
<td>IsOpen property</td>
<td>Checks whether the cursor schema is currently open.</td>
</tr>
<tr>
<td>Name property</td>
<td>Returns the name of the table.</td>
</tr>
<tr>
<td>PrimaryKey property</td>
<td>Returns the index schema of the primary key for the table.</td>
</tr>
<tr>
<td>UploadUnchangedRows property</td>
<td>Checks whether the database uploads rows that have not changed.</td>
</tr>
</tbody>
</table>

**Remarks**

There is no constructor for this class. A ULTableSchema object is attached to a table as its ULTable.Schema property.

**See also**

- “ULTableSchema class [UltraLite.NET]” on page 421
- “ULTable.Schema property [UltraLite.NET]” on page 421
- “ULCursorSchema class [UltraLite.NET]” on page 197

**GetColumnDefaultValue method**

Returns the default value of the specified column.
Visual Basic syntax

Public Function GetColumnDefaultValue(
    ByVal columnID As Integer
) As String

C# syntax

public string GetColumnDefaultValue(int columnID)

Parameters

- **columnID**  The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in a table has an ID value of zero.

Returns

The default value of the specified column as a string or a null reference (Nothing in Visual Basic) if the default value is null.

Exceptions

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

See also

- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

GetColumnPartitionSize method

Returns the global autoincrement partition size assigned to the specified column.

Visual Basic syntax

Public Function GetColumnPartitionSize(
    ByVal columnID As Integer
) As ULong

C# syntax

public ulong GetColumnPartitionSize(int columnID)

Parameters

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

Returns

The column's global autoincrement partition size as a System.UInt64 structure.

Exceptions

- **ULException class**  A SQL error occurred.
Remarks
All global autoincrement columns in a given table share the same global autoincrement partition.

See also
● “ULTableSchema.IsColumnGlobalAutoIncrement method [UltraLite.NET]” on page 430
● “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203
● System.UInt64

GetIndex method
Returns the index schema of the named index.

Visual Basic syntax
Public Function GetIndex(ByVal name As String) As ULIndexSchema

C# syntax
public ULIndexSchema GetIndex(string name)

Parameters
● name The name of the index.

Returns
A ULIndexSchema object representing the named index.

Exceptions
● ULException class A SQL error occurred.

See also
● “ULIndexSchema class [UltraLite.NET]” on page 290

GetIndexName method
Returns the name of the index identified by the specified index ID.

Visual Basic syntax
Public Function GetIndexName(ByVal indexID As Integer) As String

C# syntax
public string GetIndexName(int indexID)

Parameters
● indexID The ID of the index. The value must be in the range [1,IndexCount].
Returns
The name of the index as a string.

Exceptions
- ULException class  A SQL error occurred.

Remarks
Index IDs and counts may change during a schema upgrade. To correctly identify an index, access it by
name or refresh the cached IDs and counts after a schema upgrade.

See also
- “ULTableSchema.IndexCount property [UltraLite.NET]” on page 432

GetOptimalIndex method
The optimal index for searching a table using the specified column.

Visual Basic syntax
Public Function GetOptimalIndex(ByVal columnID As Integer) As String

C# syntax
public string GetOptimalIndex(int columnID)

Parameters
- columnID  The ID number of the column. The first column in the table has an ID value of zero.

Returns
A ULIndexSchema object representing the optimal index for the specified column.

Exceptions
- ULException class  A SQL error occurred.

Remarks
The specified column is the first column in the index, but the index may have more than one column.

See also
- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203
- “ULIndexSchema class [UltraLite.NET]” on page 290

GetPublicationPredicate method
Returns the publication predicate for this table in the named publication.
Visual Basic syntax

Public Function GetPublicationPredicate (ByVal pubName As String) As String

C# syntax

public string GetPublicationPredicate(string pubName)

Parameters

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

Returns

The publication predicate as a string.

Exceptions

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

IsColumnAutoIncrement method

Checks whether the specified column's default is set to autoincrement.

Visual Basic syntax

Public Function IsColumnAutoIncrement (ByVal columnID As Integer) As Boolean

C# syntax

public bool IsColumnAutoIncrement(int columnID)

Parameters

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

Returns

True if the column is autoincrementing, false if it is not autoincrementing.

Exceptions

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

See also

- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203
IsColumnCurrentDate method

Checks whether the specified column's default is set to the current date (a ULDbType.Date value).

Visual Basic syntax

Public Function IsColumnCurrentDate(
    ByVal columnID As Integer
) As Boolean

C# syntax

public bool IsColumnCurrentDate(int columnID)

Parameters

● columnID   The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

True if the column defaults to the current date, false if the column does not default to the current date.

Exceptions

● ULErrorException class   A SQL error occurred.

See also

● “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

IsColumnCurrentTime method

Checks whether the specified column's default is set to the current time (a ULDbType.Time value).

Visual Basic syntax

Public Function IsColumnCurrentTime(
    ByVal columnID As Integer
) As Boolean

C# syntax

public bool IsColumnCurrentTime(int columnID)

Parameters

● columnID   The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns

True if the column defaults to the current time, false if the column does not default to the current time.
Exceptions

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

See also

- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

**IsColumnCurrentTimestamp method**

Checks whether the specified column's default is set to the current timestamp (a ULDbType.TimeStamp value).

Visual Basic syntax

```vbnet
Public Function IsColumnCurrentTimestamp(  
    ByVal columnID As Integer  
) As Boolean
```

C# syntax

```csharp
public bool IsColumnCurrentTimestamp(int columnID)
```

Parameters

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

Returns

True if the column defaults to the current timestamp, false if the column does not default to the current timestamp.

Exceptions

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

See also

- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

**IsColumnCurrentUTCTimestamp method**

Checks whether the specified column's default is set to the current UTC timestamp (a ULDbType.TimeStamp value).

Visual Basic syntax

```vbnet
Public Function IsColumnCurrentUTCTimestamp(  
    ByVal columnID As Integer  
) As Boolean
```

C# syntax

```csharp
public bool IsColumnCurrentUTCTimestamp(int columnID)
```
Parameters

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

Returns

True if the column defaults to the current UTC timestamp, false if the column does not default to the current UTC timestamp.

Exceptions

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

See also

- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

**IsColumnGlobalAutoIncrement method**

Checks whether the specified column's default is set to global autoincrementing.

**Visual Basic syntax**

```vbnet
Public Function IsColumnGlobalAutoIncrement( ByVal columnID As Integer ) As Boolean
```

**C# syntax**

```csharp
public bool IsColumnGlobalAutoIncrement(int columnID)
```

Parameters

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

Returns

True if the column is global autoincrementing, false if it is not global autoincrementing.

Exceptions

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

See also

- “ULTableSchema.GetColumnPartitionSize method [UltraLite.NET]” on page 424
- “ULConnection.DatabaseID property [UltraLite.NET]” on page 156
- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203

**IsColumnNewUUID method**

Checks whether the specified column's default is set to a new UUID (a System.Guid value).
Visual Basic syntax
Public Function IsColumnNewUUID(ByVal columnID As Integer) As Boolean

C# syntax
public bool IsColumnNewUUID(int columnID)

Parameters
- columnID The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns
True if the column defaults to a new UUID, false if the column does not default to a new UUID.

Exceptions
- ULException class A SQL error occurred.

See also
- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203
- System.Guid

IsColumnNullable method
Checks whether the specified column is nullable.

Visual Basic syntax
Public Function IsColumnNullable(ByVal columnID As Integer) As Boolean

C# syntax
public bool IsColumnNullable(int columnID)

Parameters
- columnID The ID number of the column. The value must be in the range [0,ULCursorSchema.ColumnCount-1]. The first column in the table has an ID value of zero.

Returns
True if the column is nullable, false if it is not nullable.

Exceptions
- ULException class A SQL error occurred.

See also
- “ULCursorSchema.ColumnCount property [UltraLite.NET]” on page 203
**IsInPublication method**

Checks whether the table is contained in the named publication.

**Visual Basic syntax**

Public Function IsInPublication(ByVal pubName As String) As Boolean

**C# syntax**

public bool IsInPublication(string pubName)

**Parameters**

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

**Returns**

True if the table is in the publication, false if the table is not in the publication.

**Exceptions**

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

---

**IndexCount property**

Returns the number of indexes on the table.

**Visual Basic syntax**

Public ReadOnly Property IndexCount As Integer

**C# syntax**

public int IndexCount {get;}

**Remarks**

The number of indexes on the table or 0 if the table schema is closed.

Index IDs range from 1 to the IndexCount value, inclusively.

**Note**

Index IDs and count may change during a schema upgrade. To correctly identify an index, access it by name or refresh the cached IDs and counts after a schema upgrade.

---

**IsNeverSynchronized property**

Checks whether the table is marked as never being synchronized.

**Visual Basic syntax**

Public ReadOnly Property IsNeverSynchronized As Boolean
**C# syntax**

    public bool IsNeverSynchronized { get; }

**Exceptions**

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

**Remarks**

True if the table is marked as never being synchronized, false otherwise.

Tables marked as never being synchronized are never synchronized, even if they are included in a publication. These tables are sometimes referred to as "no sync" tables.

**Name property**

Returns the name of the table.

**Visual Basic syntax**

    Public ReadOnly Overrides Property Name As String

**C# syntax**

    public override string Name { get; }

**Exceptions**

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

**Remarks**

The name of the table as a string.

**PrimaryKey property**

Returns the index schema of the primary key for the table.

**Visual Basic syntax**

    Public ReadOnly Property PrimaryKey As ULIndexSchema

**C# syntax**

    public ULIndexSchema PrimaryKey { get; }

**Exceptions**

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

**Remarks**

A ULIndexSchema object representing the primary key for the table.
See also

- “ULIndexSchema class [UltraLite.NET]” on page 290

**UploadUnchangedRows property**

Checks whether the database uploads rows that have not changed.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property UploadUnchangedRows As Boolean
```

**C# syntax**

```csharp
public bool UploadUnchangedRows {get;}
```

**Exceptions**

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

**Remarks**

True if the table is marked to always upload all rows during synchronization, false if the table is marked to upload only changed rows.

Tables marked as such upload unchanged rows, as well as changed rows, when the table is synchronized. These tables are sometimes referred to as "all sync" tables.

**ULTransaction class**

Represents a SQL transaction.

**Visual Basic syntax**

```vbnet
Public NotInheritable Class ULTransaction
```

**C# syntax**

```csharp
```

**Base classes**

- **System.Data.Common.DbTransaction**

**Members**

All members of the ULTransaction class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit method</td>
<td>Commits the database transaction.</td>
</tr>
</tbody>
</table>
**Name**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose method</td>
<td>Releases the unmanaged resources used by the System.Data.Common.DbTransaction.</td>
</tr>
<tr>
<td>Rollback method</td>
<td>Rolls back the transaction's outstanding changes to the database.</td>
</tr>
<tr>
<td>Connection property</td>
<td>Returns the connection associated with the transaction.</td>
</tr>
<tr>
<td>IsolationLevel property</td>
<td>Returns the isolation level for the transaction.</td>
</tr>
</tbody>
</table>

**Remarks**

There is no constructor for the ULTransaction class. To obtain a ULTransaction object, use the ULConnection.BeginTransaction method. To associate a command with a transaction, use the ULCommand.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.

**See also**

- “ULConnection.BeginTransaction method [UltraLite.NET]” on page 125
- “ULCommand.Transaction property [UltraLite.NET]” on page 106
- System.Data.IDbTransaction
- System.IDisposable

**Commit method**

Commits the database transaction.

**Visual Basic syntax**

```
Public Overrides Sub Commit()
```

**C# syntax**

```
public override void Commit()
```

**Remarks**

Once a transaction has been committed or rolled back, the connection reverts to automatically committing all operations as they are executed. To group more operations together, a new transaction must be created.

If a Commit method call fails due to a database error (for example, a referential integrity error), the transaction remains active. Correct the error and call the Commit method again or call the ULTransaction.Rollback method to complete the transaction.
Rollback method

Rolls back the transaction's outstanding changes to the database.

Visual Basic syntax

Public Overrides Sub Rollback ()

C# syntax

public override void Rollback ()

Remarks

Once a transaction has been committed or rolled back, the connection reverts to automatically committing all operations as they are executed. To group more operations together, a new transaction must be created.

See also

● “ULTransaction.Rollback method [UltraLite.NET]” on page 436

Connection property

Returns the connection associated with the transaction.

Visual Basic syntax

Public ReadOnly Shadows Property Connection As ULConnection

C# syntax

public new ULConnection Connection {get;}

Remarks

The ULConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.


See also

● “ULConnection.BeginTransaction method [UltraLite.NET]” on page 125
● “ULConnection class [UltraLite.NET]” on page 118
● System.Data.IDbTransaction.Connection
IsolationLevel property

Returns the isolation level for the transaction.

Visual Basic syntax

Public ReadOnly Overrides Property IsolationLevel As IsolationLevel

C# syntax

public override IsolationLevel IsolationLevel {get;}

Remarks


See also

● “ULConnection.BeginTransaction method [UltraLite.NET]” on page 125
● System.Data.IsolationLevel

ULInfoMessageEventHandler delegate

Represents the method that handles the ULConnection.InfoMessage event.

Visual Basic syntax

Public Delegate Sub ULInfoMessageEventHandler(  ByVal obj As Object,  ByVal args As ULInfoMessageEventArgs)

C# syntax

public delegate void ULInfoMessageEventHandler (  object obj,  ULInfoMessageEventArgs args);

Parameters

● obj The connection sending the event.
● args The ULInfoMessageEventArgs object that contains the event data.

See also

● “ULConnection.InfoMessage event [UltraLite.NET]” on page 161
● “ULInfoMessageEventArgs class [UltraLite.NET]” on page 297
ULRowUpdatedEventHandler delegate

Represents the method that handles the ULDataAdapter.RowUpdated event.

Visual Basic syntax

Public Delegate Sub ULRowUpdatedEventHandler ( ByVal sender As Object, ByVal e As ULRowUpdatedEventArgs )

C# syntax

public delegate void ULRowUpdatedEventHandler ( object sender, ULRowUpdatedEventArgs e );

Parameters

- sender The connection sending the event.
- e The ULRowUpdatedEventArgs object that contains the event data.

See also

- “ULDataAdapter.RowUpdated event [UltraLite.NET]” on page 213
- “ULRowUpdatedEventArgs class [UltraLite.NET]” on page 366

ULRowUpdatingEventHandler delegate

Represents the method that handles the ULDataAdapter.RowUpdating event.

Visual Basic syntax

Public Delegate Sub ULRowUpdatingEventHandler ( ByVal sender As Object, ByVal e As ULRowUpdatingEventArgs )

C# syntax

public delegate void ULRowUpdatingEventHandler ( object sender, ULRowUpdatingEventArgs e );

Parameters

- sender The connection sending the event.
- e The ULRowUpdatingEventArgs object that contains the event data.
ULRowsCopiedEventHandler delegate

Represents the method that handles the ULBulkCopy.ULRowsCopied event.

**Visual Basic syntax**

```vbnet
Public Delegate Sub ULRowsCopiedEventHandler (      ByVal sender As Object,      ByVal rowsCopiedEventArgs As ULRowsCopiedEventArgs
)```

**C# syntax**

```csharp
public delegate void ULRowsCopiedEventHandler(      object sender,      ULRowsCopiedEventArgs rowsCopiedEventArgs
);
```

**Remarks**

The ULRowsCopiedEventHandler delegate is not available in the .NET Compact Framework 2.0.

**See also**

- “ULBulkCopy.ULRowsCopied event [UltraLite.NET]” on page 56
- System.Object

ULSyncProgressedDlg delegate

Represents the method that is invoked during synchronization with synchronization progress information.

**Visual Basic syntax**

```vbnet
Public Delegate Sub ULSyncProgressedDlg(      ByVal result As IAsyncResult,      ByVal data As ULSyncProgressData
)```

**C# syntax**

```csharp
public delegate void ULSyncProgressedDlg(      IAsyncResult result,      ULSyncProgressData data
);
```

**Parameters**

- **result**  The IAsyncResult object returned from the BeginSynchronize method. Use result.AsyncState to access the object provided to the BeginSynchronize method.
● data A ULSyncProgressData object containing the latest synchronization progress data.

Remarks
It is safe to do GUI work or to make UltraLite.NET API calls in this method. The synchronization is not being held up during calls to this method.

See also
● “ULConnection.BeginSynchronize method [UltraLite.NET]” on page 123
● “ULSyncProgressData class [UltraLite.NET]” on page 386

ULAuthStatusCode enumeration

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

Visual Basic syntax
Public Enum ULAuthStatusCode

C# syntax
public enum ULAuthStatusCode

Members

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

See also
● “ULSyncResult.AuthStatus property [UltraLite.NET]” on page 398
ULBulkCopyOptions enumeration

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

Visual Basic syntax

Public Enum ULBulkCopyOptions

C# syntax

public enum ULBulkCopyOptions

Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Specifying only this causes the default behavior to be used.</td>
<td>0x0</td>
</tr>
<tr>
<td>KeepIdentity</td>
<td>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.</td>
<td>0x1</td>
</tr>
<tr>
<td>UseInternal-Transaction</td>
<td>When specified, each batch of the bulk-copy operation is executed within a transaction. When not specified, transactions aren't used. If you indicate this option and also provide a ULTransaction object to the constructor, a System.ArgumentException occurs.</td>
<td>0x2</td>
</tr>
</tbody>
</table>

Remarks

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

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

See also

- “ULBulkCopy class [UltraLite.NET]” on page 46

ULDBValid enumeration

Enumerates the UltraLite.NET database validation methods

Visual Basic syntax

Public Enum ULDBValid

C# syntax

public enum ULDBValid
Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPRESS_VALIDATE</td>
<td>Performs a faster, though less thorough, validation.</td>
<td>0</td>
</tr>
<tr>
<td>FULL_VALIDATE</td>
<td>Validates tables, indexes, and all database pages.</td>
<td>1</td>
</tr>
</tbody>
</table>

See also

- “ULDatabaseManager.ValidateDatabase method [UltraLite.NET]” on page 220
- “ULConnection.ValidateDatabase method [UltraLite.NET]” on page 153

**ULDateOrder enumeration**

**UL Ext:** Enumerates the date orders that a database can support.

**Visual Basic syntax**

```vbnet
Public Enum ULDateOrder
```

**C# syntax**

```csharp
public enum ULDateOrder
```

Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YMD</td>
<td>The year followed by month, followed by day of the month.</td>
</tr>
<tr>
<td>MDY</td>
<td>The month followed by day of the month, followed by year.</td>
</tr>
<tr>
<td>DMY</td>
<td>The day of the month followed by month, followed by year.</td>
</tr>
</tbody>
</table>

**ULDbType enumeration**

Enumerates the UltraLite.NET database data types.

**Visual Basic syntax**

```vbnet
Public Enum ULDbType
```

**C# syntax**

```csharp
public enum ULDbType
```
## ULDbType enumeration

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInt</td>
<td>Signed 64-bit integer.</td>
<td>5</td>
</tr>
<tr>
<td>Binary</td>
<td>Binary data, with a specified maximum length.</td>
<td>15</td>
</tr>
<tr>
<td>Bit</td>
<td>1-bit flag.</td>
<td>8</td>
</tr>
<tr>
<td>Char</td>
<td>Character data, with a specified length.</td>
<td>0</td>
</tr>
<tr>
<td>Date</td>
<td>Date information.</td>
<td>10</td>
</tr>
<tr>
<td>DateTime</td>
<td>Timestamp information (date, time).</td>
<td>9</td>
</tr>
<tr>
<td>Decimal</td>
<td>Exact numerical data, with a specified precision and scale.</td>
<td>14</td>
</tr>
<tr>
<td>Double</td>
<td>Double precision floating-point number (8 bytes).</td>
<td>12</td>
</tr>
<tr>
<td>Float</td>
<td>Single precision floating-point number (4 bytes).</td>
<td>13</td>
</tr>
<tr>
<td>Integer</td>
<td>Unsigned 32-bit integer.</td>
<td>1</td>
</tr>
<tr>
<td>LongBinary</td>
<td>Binary data, with variable length.</td>
<td>18</td>
</tr>
<tr>
<td>LongVarChar</td>
<td>Character data, with variable length.</td>
<td>17</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Numeric</td>
<td>Exact numerical data, with a specified precision and scale.</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>The Decimal and Numeric enumeration values are aliases of each other.</td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>Single precision floating-point number (4 bytes).</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>The Float and Real enumeration values are aliases of each other.</td>
<td></td>
</tr>
<tr>
<td>SmallInt</td>
<td>Signed 16-bit integer.</td>
<td>3</td>
</tr>
<tr>
<td>STGeometry</td>
<td>ST_Geometry information.</td>
<td>ULNET_TYPE_ST_GEOMETRY</td>
</tr>
<tr>
<td>Time</td>
<td>Time information.</td>
<td>11</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Timestamp information (date, time).</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>The DateTime and TimeStamp enumeration values are aliases of each other.</td>
<td></td>
</tr>
<tr>
<td>TimeStamp-</td>
<td>Timestamp information (date, time) along with the time zone offset.</td>
<td>ULNET_TYPE_TIME-</td>
</tr>
<tr>
<td>WithTime-</td>
<td></td>
<td>STAMP_WITH_TIME_ZONE</td>
</tr>
<tr>
<td>Zone</td>
<td>TinyInt</td>
<td>7</td>
</tr>
<tr>
<td>UniqueIdent-</td>
<td>Universally Unique Identifier (UUID/GUID).</td>
<td>19</td>
</tr>
<tr>
<td>fier</td>
<td>Unsigned8-bit integer.</td>
<td></td>
</tr>
<tr>
<td>UnsignedBi-</td>
<td>Unsigned 64-bit integer.</td>
<td>6</td>
</tr>
<tr>
<td>gInt</td>
<td>Unsigned64-bit integer.</td>
<td></td>
</tr>
<tr>
<td>UnsignedInt</td>
<td>Unsigned 32-bit integer.</td>
<td>2</td>
</tr>
<tr>
<td>UnsignedS-</td>
<td>Unsigned 16-bit integer.</td>
<td>4</td>
</tr>
<tr>
<td>mallInt</td>
<td>VarBinary</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Binary data, with a specified maximum length.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Binary and VarBinary enumeration values are aliases of each other.</td>
<td></td>
</tr>
</tbody>
</table>
### Member name | Description | Value
---|---|---
VarChar | Character data, with a specified maximum length. In UltraLite.NET, this type always supports Unicode characters. The **Char** and **VarChar** types are fully compatible. | 16

**Remarks**

The table below lists which .NET types are compatible with each ULDbType. In the case of integral types, table columns can always be set using smaller integer types, but can also be set using larger types as long as the actual value is within the range of the type.

<table>
<thead>
<tr>
<th>ULDbType</th>
<th>Compatible .NET type</th>
<th>C# built-in type</th>
<th>Visual Basic built-in type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary, VarBinary</td>
<td>System.Byte[], or System.Guid if size is 16</td>
<td>byte[]</td>
<td>Byte()</td>
</tr>
<tr>
<td>Bit</td>
<td>System.Boolean</td>
<td>bool</td>
<td>Boolean</td>
</tr>
<tr>
<td>Char, VarChar</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Date</td>
<td>System.DateTime</td>
<td>DateTime (no built-in type)</td>
<td>Date</td>
</tr>
<tr>
<td>Double</td>
<td>System.Double</td>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>LongBinary</td>
<td>System.Byte[]</td>
<td>byte[]</td>
<td>Byte()</td>
</tr>
<tr>
<td>LongVarChar</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Decimal, Numeric</td>
<td>System.Decimal</td>
<td>decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>Float, Real</td>
<td>System.Single</td>
<td>float</td>
<td>Single</td>
</tr>
<tr>
<td>BigInt</td>
<td>System.Int64</td>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>Integer</td>
<td>System.Int32</td>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>SmallInt</td>
<td>System.Int16</td>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>ST_Geometry</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Time</td>
<td>System.TimeSpan</td>
<td>TimeSpan (no built-in type)</td>
<td>TimeSpan (no built-in type)</td>
</tr>
<tr>
<td>ULDbType</td>
<td>Compatible .NET type</td>
<td>C# built-in type</td>
<td>Visual Basic built-in type</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>DateTime, TimeStamp</td>
<td>System.DateTime</td>
<td>DateTime (no built-in type)</td>
<td>Date</td>
</tr>
<tr>
<td>TimeStampWithTimeZone</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>TinyInt</td>
<td>System.Byte</td>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>UnsignedBigInt</td>
<td>System.UInt64</td>
<td>ulong</td>
<td>UInt64 (no built-in type)</td>
</tr>
<tr>
<td>UnsignedInt</td>
<td>System.UInt32</td>
<td>uint</td>
<td>UInt32 (no built-in type)</td>
</tr>
<tr>
<td>UnsignedSmallInt</td>
<td>System.UInt16</td>
<td>ushort</td>
<td>UInt16 (no built-in type)</td>
</tr>
<tr>
<td>UniqueIdentifier</td>
<td>System.Guid</td>
<td>Guid (no built-in type)</td>
<td>Guid (no built-in type)</td>
</tr>
</tbody>
</table>

Binary columns of length 16 are fully compatible with the UniqueIdentifier type.

See also

- “ULDataReader.GetFieldType method [UltraLite.NET]” on page 243
- “ULDataReader.GetDataTypeName method [UltraLite.NET]” on page 240
- “ULCursorSchema.GetColumnULDbType method [UltraLite.NET]” on page 202
- System.Byte
- System.Guid
- System.Boolean
- System.String
- System.DateTime
- System.Single
- System.Int64
- System.Int32
- System.Int16
- System.TimeSpan
- System.UInt64
- System.UInt32
- System.UInt16

**ULRuntimeType enumeration**

**UL Ext:** Enumerates the types of UltraLite.NET runtimes.
Visual Basic syntax
   Public Enum ULRuntimeType

C# syntax
   public enum ULRuntimeType

Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAND-ALONE_UL</td>
<td>Selects the standalone UltraLite.NET runtime. The standalone runtime accesses databases directly. Databases are accessed more quickly this way, but cannot be shared.</td>
<td>0</td>
</tr>
<tr>
<td>UL_ENGINE_CLIENT</td>
<td>Selects the UltraLite engine runtime. The UltraLite.NET engine client communicates with the UltraLite engine to access databases. Databases can be shared by different applications.</td>
<td>1</td>
</tr>
</tbody>
</table>

See also
- “ULDatabaseManager.RuntimeType property [UltraLite.NET]” on page 220

ULSqlProgressState enumeration

UL Ext: Enumerates all the states that can occur while executing SQL passthrough scripts.

Visual Basic syntax
   Public Enum ULSqlProgressState

C# syntax
   public enum ULSqlProgressState

Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE_STARTING</td>
<td>No scripts have been executed yet.</td>
<td>0</td>
</tr>
<tr>
<td>STATE_RUNNING_SCRIPT</td>
<td>Currently running a SQL passthrough script.</td>
<td>1</td>
</tr>
<tr>
<td>STATE_DONE</td>
<td>Scripts have successfully completed.</td>
<td>2</td>
</tr>
<tr>
<td>STATE_ERROR</td>
<td>Scripts have completed, but an error occurred.</td>
<td>3</td>
</tr>
</tbody>
</table>
ULStreamType enumeration

UL Ext: Enumerates the types of MobiLink synchronization streams to use for synchronization.

Visual Basic syntax

    Public Enum ULStreamType

C# syntax

    public enum ULStreamType

Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPIP</td>
<td>Synchronize via TCP/IP.</td>
</tr>
<tr>
<td>HTTP</td>
<td>Synchronize via HTTP.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Synchronize via HTTPS (HTTP with transport-layer security).</td>
</tr>
<tr>
<td>TLS</td>
<td>Synchronize via TCP/IP with transport layer security.</td>
</tr>
</tbody>
</table>

Remarks

Note
Separately licensed component required.

FIPS-certified encryption requires a separate license. All strong encryption technologies are subject to export regulations.

See “Separately licensed components” [SQL Anywhere 16 - Introduction].

See also

- “ULSyncParms.Stream property [UltraLite.NET]” on page 384
- “UltraLite network protocol options” [UltraLite - Database Management and Reference]
ULSyncProgressState enumeration

UL Ext: Enumerates all the states that can occur while synchronizing.

Visual Basic syntax

```vbnet
Public Enum ULSyncProgressState
```

C# syntax

```csharp
public enum ULSyncProgressState
```

Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE_STARTING</td>
<td>No synchronization actions have been taken yet.</td>
<td>0</td>
</tr>
<tr>
<td>STATE_CONNECTING</td>
<td>The synchronization stream has been built, but is not yet opened.</td>
<td>1</td>
</tr>
<tr>
<td>STATE_SENDING_HEADER</td>
<td>The synchronization stream has been opened and the header is about to be sent.</td>
<td>2</td>
</tr>
<tr>
<td>STATE_SENDING_TABLE</td>
<td>A table is being sent. Progress can be monitored using the ULSyncProgressData.SyncTableIndex and ULSyncProgressData.SyncTableCount properties.</td>
<td>3</td>
</tr>
<tr>
<td>STATE_SENDING_DATA</td>
<td>Data for the current table is being sent.</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>The ULSyncProgressData.SentBytes, ULSyncProgressData.SentInserts, ULSyncProgressData.SentUpdates, and ULSyncProgressData.SentDeletes properties have been updated.</td>
<td></td>
</tr>
<tr>
<td>STATE_FINISHING_UPLOAD</td>
<td>The upload is completing.</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>The final count of rows sent is included with this event.</td>
<td></td>
</tr>
<tr>
<td>STATE RECEIVING UPLOAD ACK</td>
<td>An acknowledgement that the upload is complete is being received.</td>
<td>6</td>
</tr>
<tr>
<td>STATE RECEIVING TABLE</td>
<td>A table is being received. Progress can be monitored using the ULSyncProgressData.SyncTableIndex and ULSyncProgressData.SyncTableCount properties.</td>
<td>7</td>
</tr>
<tr>
<td><strong>Member name</strong></td>
<td><strong>Description</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>STATE RECEIVING DATA</td>
<td>Data for the current table is being received. The ULSyncProgressData.ReceivedBytes, ULSyncProgressData.ReceivedInserts, ULSyncProgressData.ReceivedUpdates, and ULSyncProgressData.ReceivedDeletes properties have been updated.</td>
<td>8</td>
</tr>
<tr>
<td>STATE COMMITTING DOWNLOAD</td>
<td>The download is being committed. The final count of rows received is included with this event.</td>
<td>9</td>
</tr>
<tr>
<td>STATE ROLLING_BACK_DOWNLOAD</td>
<td>Synchronization is rolling back the download because an error was encountered during the download. The error is reported with a subsequent STATE_ERROR progress report.</td>
<td>10</td>
</tr>
<tr>
<td>STATE_SENDING_DOWNLOAD_ACK</td>
<td>An acknowledgement of download completion is being sent.</td>
<td>11</td>
</tr>
<tr>
<td>STATE_DISCONNECTING</td>
<td>The synchronization stream is about to be closed.</td>
<td>12</td>
</tr>
<tr>
<td>STATE_DONE</td>
<td>Synchronization has successfully completed.</td>
<td>13</td>
</tr>
<tr>
<td>STATE_ERROR</td>
<td>Synchronization has completed, but an error occurred.</td>
<td>14</td>
</tr>
<tr>
<td>STATE_CANCELLED</td>
<td>Synchronization has been canceled.</td>
<td>15</td>
</tr>
</tbody>
</table>

**See also**

- “ULSyncProgressData class [UltraLite.NET]” on page 386
- “ULSyncProgressData.ReceivedBytes property [UltraLite.NET]” on page 390
- “ULSyncProgressData.ReceivedInserts property [UltraLite.NET]” on page 390
- “ULSyncProgressData.ReceivedUpdates property [UltraLite.NET]” on page 391
- “ULSyncProgressData.ReceivedDeletes property [UltraLite.NET]” on page 390
- “ULSyncProgressData.SentBytes property [UltraLite.NET]” on page 391
- “ULSyncProgressData.SentInserts property [UltraLite.NET]” on page 392
- “ULSyncProgressData.SentUpdates property [UltraLite.NET]” on page 392
- “ULSyncProgressData.SentDeletes property [UltraLite.NET]” on page 392
- “ULSyncProgressData.SyncTableIndex property [UltraLite.NET]” on page 393
- “ULSyncProgressData.SyncTableCount property [UltraLite.NET]” on page 393
Index

A
Abort property
ULRowsCopiedEventArgs class [UltraLite.NET API], 365
accessing data
UltraLite.NET Table API, 9
accessing schema information
UltraLite.NET about, 16
ActiveSync synchronization
UltraLite.NET, 19
ActiveSyncInvoked method
ULActiveSyncListener interface [UltraLite.NET API], 44
Add method
ULBulkCopyColumnMappingCollection class
[UltraLite.NET API], 65
ULParameterCollection class [UltraLite.NET API], 323
AdditionalParms property
ULConnectionParms class [UltraLite.NET API], 167
ULSyncParms class [UltraLite.NET API], 377
AddRange method
ULParameterCollection class [UltraLite.NET API], 329
AppendBytes method
ULResultSet class [UltraLite.NET API], 345
AppendChars method
ULResultSet class [UltraLite.NET API], 347
applications
developing for Windows Mobile, 3
architectures
UltraLite.NET, 43
AuthenticationParms property
ULFileTransfer class [UltraLite.NET API], 279
ULSyncParms class [UltraLite.NET API], 378
AuthStatus property
ULFileTransfer class [UltraLite.NET API], 280
ULSyncResult class [UltraLite.NET API], 398
AuthValue property
ULFileTransfer class [UltraLite.NET API], 280
ULSyncResult class [UltraLite.NET API], 398
AutoCommit mode
UltraLite.NET development, 15

B
BatchSize property
ULBulkCopy class [UltraLite.NET API], 53
BeginExecuteNonQuery method
ULCommand class [UltraLite.NET API], 78
BeginExecuteReader method
ULCommand class [UltraLite.NET API], 80
BeginSynchronize method
ULConnection class [UltraLite.NET API], 123
BeginTransaction method
ULConnection class [UltraLite.NET API], 125
BulkCopyTimeout property
ULBulkCopy class [UltraLite.NET API], 54
BytesReceived property
ULFileTransferProgressData class [UltraLite.NET API], 287

C
CacheSize property
ULConnectionParms class [UltraLite.NET API], 168
ULConnectionStringBuilder class [UltraLite.NET API], 180
Cancel method
ULCommand class [UltraLite.NET API], 83
CancelGetNotification method
ULConnection class [UltraLite.NET API], 128
CancelSynchronize method
ULConnection class [UltraLite.NET API], 129
CanCreateDataSourceEnumerator property
ULFactory class [UltraLite.NET API], 271
CaseSensitive property
ULCreateParms class [UltraLite.NET API], 190
casting
UltraLite.NET data types in, 15
ChangeDatabase method
ULConnection class [UltraLite.NET API], 129
ChangeEncryptionKey method
ULConnection class [UltraLite.NET API], 130
ChangePassword method
ULConnection class [UltraLite.NET API], 130
ChecksumLevel property
ULSyncResult class [UltraLite.NET API], 398
Clear method
ULParameterCollection class [UltraLite.NET API], 330
Close method
data manipulation
  UltraLite.NET Table API, 9, 10
data modification
  UltraLite.NET with SQL, 5
data types
  UltraLite.NET API accessing and casting, 14
DataAdapter property
  ULCommandBuilder class [UltraLite.NET API], 117
Database property
  ULConnection class [UltraLite.NET API], 156
database schemas
  UltraLite.NET access, 16
DatabaseID property
  ULConnection class [UltraLite.NET API], 156
DatabaseKey property
  ULConnectionStringBuilder class [UltraLite.NET API], 181
DatabaseManager class
  UltraLite.NET, 3
DatabaseName property
  ULConnectionStringBuilder class [UltraLite.NET API], 181
DatabaseOnDesktop property
  ULConnectionParms class [UltraLite.NET API], 170
  ULConnectionStringBuilder class [UltraLite.NET API], 182
DatabaseOnDevice property
  ULConnectionParms class [UltraLite.NET API], 170
  ULConnectionStringBuilder class [UltraLite.NET API], 183
DataSource property
  ULConnection class [UltraLite.NET API], 157
DataSourceInformation property
  ULMetaDataCollectionNames class [UltraLite.NET API], 301
DataTypes property
  ULMetaDataCollectionNames class [UltraLite.NET API], 301
DateFormat property
  ULCreatParms class [UltraLite.NET API], 191
DateOrder property
  ULCreatParms class [UltraLite.NET API], 192
DbType property
  ULParameter class [UltraLite.NET API], 315
DeclareEvent method
  ULConnection class [UltraLite.NET API], 133
Delete method
  ULResultSet class [UltraLite.NET API], 348
DeleteAllRows method
  ULTable class [UltraLite.NET API], 407
DeleteCommand property
  ULDataAdapter class [UltraLite.NET API], 210
deleting
  UltraLite.NET table rows, 15
deploying
  UltraLite.NET applications, 19, 20, 21
Depth property
  ULReader class [UltraLite.NET API], 260
DesignTimeVisible property
  ULCommand class [UltraLite.NET API], 104
DestinationColumn property
  ULBulkCopyColumnMapping class [UltraLite.NET API], 60
DestinationOrdinal property
  ULBulkCopyColumnMapping class [UltraLite.NET API], 61
DestinationTableName property
  ULBulkCopy class [UltraLite.NET API], 55
DestroyNotificationQueue method
  ULConnection class [UltraLite.NET API], 133
development
  UltraLite.NET, 3
development platforms
  UltraLite.NET, 1
Direction property
  ULParameter class [UltraLite.NET API], 316
Dispose method
  ULBulkCopy class [UltraLite.NET API], 51
dynamic SQL
  UltraLite.NET tutorial, 25
EncryptionKey property
  ULConnectionParms class [UltraLite.NET API], 170
EndExecuteNonQuery method
  ULCommand class [UltraLite.NET API], 84
EndExecuteReader method
  ULCommand class [UltraLite.NET API], 88
EndSynchronize method
  ULConnection class [UltraLite.NET API], 134
EquivalentTo method
  ULConnectionStringBuilder class [UltraLite.NET API], 178
error handling
  UltraLite.NET, 17
errors
  UltraLite.NET API handling, 17
ExecuteNonQuery method
  ULCommand class [UltraLite.NET API], 91
ExecuteReader method
  ULCommand class [UltraLite.NET API], 92
ExecuteResultSet method
  ULCommand class [UltraLite.NET API], 95
ExecuteScalar method
  ULCommand class [UltraLite.NET API], 97
ExecuteTable method
  ULCommand class [UltraLite.NET API], 98
ULConnection class [UltraLite.NET API], 135
FieldCount property
  ULDataReader class [UltraLite.NET API], 260
FileAuthCode property
  ULFileTransfer class [UltraLite.NET API], 280
FileName property
  ULFileTransfer class [UltraLite.NET API], 281
FileSize property
  ULFileTransferProgressData class [UltraLite.NET API], 288
FileTransferProgressed method
  ULFileTransferProgressListener interface [UltraLite.NET API], 289
find methods
  UltraLite.NET, 13
find mode
  UltraLite.NET, 11
FindBegin method
GetColumnScale method
  ULCursorSchema class [UltraLite.NET API], 200
GetColumnSize method
  ULCursorSchema class [UltraLite.NET API], 201
GetColumnSQLName method
  ULCursorSchema class [UltraLite.NET API], 201
GetColumnULDbType method
  ULCursorSchema class [UltraLite.NET API], 202
GetDatabaseProperty method
  ULDatabaseSchema class [UltraLite.NET API], 222
GetDataTypeName method
  ULDataReader class [UltraLite.NET API], 240
GetDateTime method
  ULDataReader class [UltraLite.NET API], 240
GetDecimal method
  ULDataReader class [UltraLite.NET API], 241
GetDeleteCommand method
  ULCommandBuilder class [UltraLite.NET API], 111
GetDouble method
  ULDataReader class [UltraLite.NET API], 242
GetEnumerator method
  ULDataReader class [UltraLite.NET API], 242
ULParameterCollection class [UltraLite.NET API], 332
GetFloat method
  ULDataReader class [UltraLite.NET API], 243
GetGuid method
  ULDataReader class [UltraLite.NET API], 244
GetIndex method
  ULTableSchema class [UltraLite.NET API], 425
GetIndexName method
  ULTableSchema class [UltraLite.NET API], 425
GetInsertCommand method
  ULCommandBuilder class [UltraLite.NET API], 113
GetInt16 method
  ULDataReader class [UltraLite.NET API], 244
GetInt32 method
  ULDataReader class [UltraLite.NET API], 245
GetInt64 method
  ULDataReader class [UltraLite.NET API], 246
GetLastDownloadTime method
  ULConnection class [UltraLite.NET API], 139
GetName method
  ULDataReader class [UltraLite.NET API], 246
GetNewUUID method
  ULConnection class [UltraLite.NET API], 140
GetNotification method
  ULConnection class [UltraLite.NET API], 140
GetNotificationParameter method
  ULConnection class [UltraLite.NET API], 141
GetObjectData method
  ULException class [UltraLite.NET API], 266
GetOptimalIndex method
  ULTableSchema class [UltraLite.NET API], 426
GetOrdinal method
  ULDataReader class [UltraLite.NET API], 247
GetPublicationName method
  ULDatabaseSchema class [UltraLite.NET API], 224
GetPublicationPredicate method
  ULTableSchema class [UltraLite.NET API], 426
GetRowCount method
  ULDataReader class [UltraLite.NET API], 248
GetSchema method
  ULConnection class [UltraLite.NET API], 142
GetSchemaTable method
  ULCursorSchema class [UltraLite.NET API], 203
ULDataReader class [UltraLite.NET API], 249
GetShortName method
  ULConnectionStringBuilder class [UltraLite.NET API], 178
GetString method
  ULDataReader class [UltraLite.NET API], 250
GetTableName method
  ULDatabaseSchema class [UltraLite.NET API], 225
GetTimeSpan method
  ULDataReader class [UltraLite.NET API], 251
GetUInt16 method
  ULDataReader class [UltraLite.NET API], 252
GetUInt32 method
  ULDataReader class [UltraLite.NET API], 252
GetUInt64 method
  ULDataReader class [UltraLite.NET API], 253
GetUpdateCommand method
  ULCommandBuilder class [UltraLite.NET API], 115
GetValue method
  ULDataReader class [UltraLite.NET API], 254
GetValues method
  UDataReader class [UltraLite.NET API], 254
GlobalAutoIncrementUsage property
  ULConnection class [UltraLite.NET API], 157
GrantConnectTo method
  ULConnection class [UltraLite.NET API], 145

HasRows property
  UDataReader class [UltraLite.NET API], 260

iAnywhere.Data.UltraLite namespace
  UltraLite.NET API, 43

IGNORED_DELETEES property
  ULSyncProgressData class [UltraLite.NET API], 389

IGNORED_ROWS property
  ULSyncResult class [UltraLite.NET API], 398

IGNORED_UPDATES property
  ULSyncProgressData class [UltraLite.NET API], 389

INDEX_COLUMNS property
  ULMetaDataCollectionNames class [UltraLite.NET API], 303

INDEX_COUNT property
  ULTTableSchema class [UltraLite.NET API], 432

INDEXES property
  ULMetaDataCollectionNames class [UltraLite.NET API], 303

INDEX_NAME property
  ULCmdExe class [UltraLite.NET API], 104

INDEX_OF method
  ULBulkCopyColumnMappingCollection class [UltraLite.NET API], 69
  ULParameterCollection class [UltraLite.NET API], 333

INFO_MESSAGE event
  ULCmdExe class [UltraLite.NET API], 161

INSERT method
  ULParameterCollection class [UltraLite.NET API], 334
  ULTTable class [UltraLite.NET API], 415

insert mode
  UltraLite.NET, 11

INSERTBEGIN method
  ULTTable class [UltraLite.NET API], 416

INSERTCOMMAND property
  ULD_column class [UltraLite.NET API], 211

inserting
  UltraLite.NET table rows, 11

Instance field
  ULCmdExe class [UltraLite.NET API], 271
  INVALID_DATABASE_ID field
    ULParameterCollection class [UltraLite.NET API], 163

IsBOF property
  UDataReader class [UltraLite.NET API], 261

IsCaseSensitive property
  ULDDatabaseSchema class [UltraLite.NET API], 227

IsClosed property
  UDataReader class [UltraLite.NET API], 261

IsColumnAutoIncrement method
  ULTTableSchema class [UltraLite.NET API], 427

IsColumnCurrentDate method
  ULTTableSchema class [UltraLite.NET API], 428

IsColumnCurrentTime method
  ULTTableSchema class [UltraLite.NET API], 428

IsColumnCurrentTimestamp method
  ULTTableSchema class [UltraLite.NET API], 429

IsColumnDescending method
  ULDIndexSchema class [UltraLite.NET API], 292

IsColumnGlobalAutoIncrement method
  ULTTableSchema class [UltraLite.NET API], 430

IsColumnNewUUID method
  ULTTableSchema class [UltraLite.NET API], 430

IsColumnNullable method
  ULTTableSchema class [UltraLite.NET API], 431

IsDBNull method
  ULDParameter class [UltraLite.NET API], 255

IsEOF property
  UDataReader class [UltraLite.NET API], 261

IsFinalSyncProgress property
  ULSyncProgressData class [UltraLite.NET API], 389

IsFixedSize property
  ULDParameterCollection class [UltraLite.NET API], 336

IsForeign property
  ULDIndexSchema class [UltraLite.NET API], 293

IsForeignCheckOnCommit property
  UltraLite.NET, 11
ULIndexSchema class [UltraLite.NET API], 293
IsForeignKeyNullable property
ULIndexSchema class [UltraLite.NET API], 294
IsInPublication method
ULTableSchema class [UltraLite.NET API], 432
IsNeverSynchronized property
ULTableSchema class [UltraLite.NET API], 432
IsNullable property
ULParameter class [UltraLite.NET API], 317
IsolationLevel property
ULTransaction class [UltraLite.NET API], 437
IsOpen property
ULCursorSchema class [UltraLite.NET API], 203
ULDatabaseSchema class [UltraLite.NET API], 227
ULIndexSchema class [UltraLite.NET API], 294
IsPrimaryKey property
ULIndexSchema class [UltraLite.NET API], 295
IsReadOnly property
ULParameterCollection class [UltraLite.NET API], 337
IsSynchronized property
ULParameterCollection class [UltraLite.NET API], 337
IsUniqueIndex property
ULIndexSchema class [UltraLite.NET API], 295
IsUniqueKey property
ULIndexSchema class [UltraLite.NET API], 295

K

KeepPartialDownload property
ULSyncParms class [UltraLite.NET API], 379

L

LastIdentity property
ULConnection class [UltraLite.NET API], 158
LocalFileName property
ULFileTransfer class [UltraLite.NET API], 281
LocalPath property
ULFileTransfer class [UltraLite.NET API], 282
locking
UltraLite.NET keywords for, 19
lookup methods
UltraLite.NET, 13
lookup mode
UltraLite.NET, 11
LookupBackward method
ULTable class [UltraLite.NET API], 416
LookupBegin method
ULTable class [UltraLite.NET API], 418
LookupForward method
ULTable class [UltraLite.NET API], 419

M

managing
UltraLite.NET transactions, 15
MaxHashSize property
ULCreateParms class [UltraLite.NET API], 193
Message property
ULInfoMessageEventArgs class [UltraLite.NET API], 298
MetaDataCollections property
ULMetaDataCollectionNames class [UltraLite.NET API], 304
modes
UltraLite.NET, 11
MoveAfterLast method
ULDataReader class [UltraLite.NET API], 256
MoveBeforeFirst method
ULDataReader class [UltraLite.NET API], 256
MoveFirst method
ULDataReader class [UltraLite.NET API], 256
UltraLite.NET data retrieval example, 8
moveFirst method (Table class)
UltraLite.NET development, 10
MoveLast method
ULDataReader class [UltraLite.NET API], 257
MoveNext method
ULDataReader class [UltraLite.NET API], 257
UltraLite.NET data retrieval example, 8
moveNext method (Table class)
UltraLite.NET development, 10
MovePrevious method
ULDataReader class [UltraLite.NET API], 258
MoveRelative method
ULDataReader class [UltraLite.NET API], 258
multi-threaded applications
UltraLite.NET, 4

N

Name property
ULCursorSchema class [UltraLite.NET API], 204
ULIndexSchema class [UltraLite.NET API], 296
ULResultSetSchema class [UltraLite.NET API], 364
ULTableSchema class [UltraLite.NET API], 433
NativeError property
ULException class [UltraLite.NET API], 266
ULInfoMessageEventArgs class [UltraLite.NET API], 299
navigating
UltraLite.NET Table API, 10
navigating SQL result sets
UltraLite.NET, 9
NearestCentury property
ULCreateParms class [UltraLite.NET API], 193
NewPassword property
ULSyncParms class [UltraLite.NET API], 380
NextResult method
ULDataReader class [UltraLite.NET API], 259
NotifyAfter property
ULBulkCopy class [UltraLite.NET API], 55

O
Obfuscate property
ULCreateParms class [UltraLite.NET API], 193
Offset property
ULParameter class [UltraLite.NET API], 317
offsets
UltraLite.NET relative, 10
Open method
ULConnection class [UltraLite.NET API], 145
OrderedTableScans property
ULConnectionStringBuilder class [UltraLite.NET API], 183

P
PageSize property
ULCreateParms class [UltraLite.NET API], 194
ParameterName property
ULParameter class [UltraLite.NET API], 317
Parameters property
ULCommand class [UltraLite.NET API], 105
PartialDownloadRetained property
ULSyncResult class [UltraLite.NET API], 399
Password property
ULConnectionParms class [UltraLite.NET API], 171
ULConnectionStringBuilder class [UltraLite.NET API], 184
ULFileSyncTransfer class [UltraLite.NET API], 282
ULSyncParms class [UltraLite.NET API], 381
PingOnly property
ULSyncParms class [UltraLite.NET API], 381
Plan property
ULCommand class [UltraLite.NET API], 105
platforms supported in UltraLite.NET, 1
Precision property
ULCreateParms class [UltraLite.NET API], 194
ULParameter class [UltraLite.NET API], 318
Prepare method
ULCommand class [UltraLite.NET API], 101
PrimaryKey property
ULTableSchema class [UltraLite.NET API], 433
PublicationCount property
ULDatabaseSchema class [UltraLite.NET API], 227
Publications property
ULMetaDataCollectionNames class
[UltraLite.NET API], 305
ULSyncParms class [UltraLite.NET API], 382

R
Read method
ULDataReader class [UltraLite.NET API], 259
ReceivedBytes property
ULSyncProgressData class [UltraLite.NET API], 390
ReceivedDeletes property
ULSyncProgressData class [UltraLite.NET API], 390
ReceivedInserts property
ULSyncProgressData class [UltraLite.NET API], 390
ReceivedUpdates property
ULSyncProgressData class [UltraLite.NET API], 391
RecordsAffected property
ULDataReader class [UltraLite.NET API], 262
ULRowUpdatedEventArgs class [UltraLite.NET API], 368
ReferencedIndexName property
ULIndexSchema class [UltraLite.NET API], 296
ReferencedTableName property
ULIndexSchema class [UltraLite.NET API], 297
RegisterForEvent method
ULSyncProgressData class [UltraLite.NET API], 393
StateChange event
ULConnection class [UltraLite.NET API], 162
StopSynchronizationDelete method
ULConnection class [UltraLite.NET API], 150
Stream property
ULFileTransfer class [UltraLite.NET API], 284
ULSyncParms class [UltraLite.NET API], 384
StreamErrorCode property
ULFileTransfer class [UltraLite.NET API], 284
ULSyncResult class [UltraLite.NET API], 399
StreamErrorParameters property
ULSyncResult class [UltraLite.NET API], 399
StreamErrorSystem property
ULFileTransfer class [UltraLite.NET API], 285
ULSyncResult class [UltraLite.NET API], 400
StreamParms property
ULFileTransfer class [UltraLite.NET API], 285
ULSyncParms class [UltraLite.NET API], 384
supported platforms
UltraLite.NET, 1
SYNC_ALL_DB field
ULConnection class [UltraLite.NET API], 163
SYNC_ALL_PUBS field
ULConnection class [UltraLite.NET API], 163
synchronization
ActiveSync in UltraLite.NET, 19
UltraLite.NET C# example, 18
Synchronize method
ULConnection class [UltraLite.NET API], 150
synchronizing
UltraLite.NET, 18
SyncParms property
ULConnection class [UltraLite.NET API], 160
SyncProgressed method
ULSyncProgressListener interface [UltraLite.NET API], 396
SyncResult property
ULConnection class [UltraLite.NET API], 160
SyncRoot property
ULParameterCollection class [UltraLite.NET API], 337
SyncTableCount property
ULSyncProgressData class [UltraLite.NET API], 393
SyncTableIndex property
ULSyncProgressData class [UltraLite.NET API], 393

T
Table API
UltraLite.NET introduction, 9
TableCount property
ULDatabaseSchema class [UltraLite.NET API], 228
TableID property
ULSyncProgressData class [UltraLite.NET API], 394
TableMappings property
ULDataAdapter class [UltraLite.NET API], 212
 TableName property
ULSyncProgressData class [UltraLite.NET API], 394
tables
UltraLite.NET schema information in, 16
Tables property
ULMetaDataCollectionNames class [UltraLite.NET API], 306
target platforms
UltraLite.NET, 1
this property
ULBulkCopyColumnMappingCollection class [UltraLite.NET API], 70
ULConnectionStringBuilder class [UltraLite.NET API], 186
ULDataReader class [UltraLite.NET API], 263
ULParameterCollection class [UltraLite.NET API], 338
threads
UltraLite.NET API multi-threaded applications, 4
TimeFormat property
ULCreateParms class [UltraLite.NET API], 195
Timestamp property
ULSyncResult class [UltraLite.NET API], 400
TimestampFormat property
ULCreateParms class [UltraLite.NET API], 196
TimestampIncrement property
ULCreateParms class [UltraLite.NET API], 196
ToString method
ULConnectionParms class [UltraLite.NET API], 167
ULCreateParms class [UltraLite.NET API], 190
ULInfoMessageEventArgs class [UltraLite.NET API], 298
ULParameter class [UltraLite.NET API], 315
ULSyncParms class [UltraLite.NET API], 377
TotalDownloadRowCount property
ULSyncProgressData class [UltraLite.NET API], 394
transaction processing
UltraLite.NET management, 15
Transaction property
ULCommand class [UltraLite.NET API], 106
transactions
UltraLite.NET management, 15
TransferredFile property
ULFileTransfer class [UltraLite.NET API], 285
TriggerEvent method
ULConnection class [UltraLite.NET API], 152
troubleshooting
UltraLite.NET deployment checklist, 25
UltraLite.NET handling errors, 17
Truncate method
ULTable class [UltraLite.NET API], 420
TruncateDeletes property
ULSyncProgressData class [UltraLite.NET API], 395
TryGetValue method
ULConnectionStringBuilder class [UltraLite.NET API], 179
tutorials
building a Windows Mobile application, 25
ULActiveSyncListener interface [UltraLite.NET API]
ActiveSyncInvoked method, 44
description, 44
ULAuthStatusCode enumeration [UltraLite.NET API]
description, 440
ULBulkCopy class [UltraLite.NET API]
BatchSize property, 53
BulkCopyTimeout property, 54
Close method, 51
columnMappings property, 54
description, 46
DestinationTableName property, 55
Dispose method, 51
NotifyAfter property, 55
ULBulkCopy constructor, 47
ULRowsCopied event, 56
WriteToServer method, 51
ULBulkCopy constructor
ULBulkCopy class [UltraLite.NET API], 47
ULBulkCopyColumnMapping class [UltraLite.NET API]
description, 56
DestinationColumn property, 60
DestinationOrdinal property, 61
SourceColumn property, 61
SourceOrdinal property, 62
ULBulkCopyColumnMapping constructor, 57
ULBulkCopyColumnMapping constructor
ULBulkCopyColumnMapping class [UltraLite.NET API], 57
ULBulkCopyColumnMappingCollection class [UltraLite.NET API]
Add method, 65
Contains method, 68
CopyTo method, 69
description, 62
IndexOf method, 69
Remove method, 70
RemoveAt method, 70
this property, 70
ULBulkCopyOptions enumeration [UltraLite.NET API]
description, 441
ULCommand class
UltraLite.NET data manipulation example, 5
UltraLite.NET data retrieval example, 8
ULCommand class [UltraLite.NET API]
BeginExecuteNonQuery method, 78
BeginExecuteReader method, 80
Cancel method, 83
CommandText property, 101
CommandTimeout property, 102
CommandType property, 102
Connection property, 103
CreateParameter method, 84
description, 71
DesignTimeVisible property, 104
EndExecuteNonQuery method, 84
EndExecuteReader method, 88
ExecuteNonQuery method, 91
ExecuteReader method, 92
ExecuteResultSet method, 95
ExecuteScalar method, 97
DatabaseName property, 181
DatabaseOnDesktop property, 182
DatabaseOnDevice property, 183
description, 172
EquivalentTo method, 178
GetShortName method, 178
OrderedTableScans property, 183
Password property, 184
Remove method, 179
ReserveSize property, 184
StartLine property, 185
this property, 186
TryGetValue method, 179
ULConnectionStringBuilder constructor, 176
UserID property, 187
ULConnectionStringBuilder constructor
ULConnectionStringBuilder class [UltraLite.NET API], 176
ULCreateParms class [UltraLite.NET API]
  CaseSensitive property, 190
  ChecksumLevel property, 191
  DateFormat property, 191
  DateOrder property, 192
description, 188
  FIPS property, 192
  MaxHashSize property, 193
  NearestCentury property, 193
  Obfuscate property, 193
  PageSize property, 194
  Precision property, 194
  Scale property, 195
  TimeFormat property, 195
  TimestampFormat property, 196
  TimestampIncrement property, 196
  ToString method, 190
  ULCreateParms constructor, 190
  UTF8Encoding property, 196
ULCreateParms constructor
  ULCreateParms class [UltraLite.NET API], 190
ULCursorSchema class [UltraLite.NET API]
  ColumnCount property, 203
description, 197
  GetColumnID method, 198
  GetColumnName method, 199
  GetColumnPrecision method, 200
  GetColumnScale method, 200
  GetColumnSize method, 201
  GetColumnSQLName method, 201
  getColumnULDbType method, 202
  GetSchemaTable method, 203
  IsOpen property, 203
  Name property, 204
ULDataAdapter class [UltraLite.NET API]
  DeleteCommand property, 210
description, 204
  GetFillParameters method, 210
  InsertCommand property, 211
  RowUpdated event, 213
  RowUpdating event, 214
  SelectCommand property, 211
  TableMappings property, 212
  ULDataAdapter constructor, 207
  UpdateCommand property, 213
ULDataAdapter constructor
  ULDataAdapter class [UltraLite.NET API], 207
ULDatabaseManager class [UltraLite.NET API]
  CreateDatabase method, 215
description, 214
  DropDatabase method, 216
  RuntimeType property, 220
  SetActiveSyncListener method, 217
  SetServerSyncListener method, 218
  SignalSyncIsComplete method, 219
  ValidateDatabase method, 220
ULDatabaseSchema class
  UltraLite.NET development, 16
ULDatabaseSchema class [UltraLite.NET API]
description, 221
  GetDatabaseProperty method, 222
  GetPublicationName method, 224
  GetTableName method, 225
  IsCaseSensitive property, 227
  IsOpen property, 227
  PublicationCount property, 227
  SetDatabaseOption method, 225
  TableCount property, 228
ULDataReader class
  UltraLite.NET data retrieval example, 8
ULDataReader class [UltraLite.NET API]
  Close method, 234
description, 228
  Depth property, 260
  FieldCount property, 260
  GetBoolean method, 234
  GetByte method, 235
  GetBytes method, 235
ResumedAtSize property, 288
ULFileTransferProgressListener interface
[UltraLite.NET API]
description, 289
FileTransferProgressed method, 289
ULIndexSchema class
UltraLite.NET development, 16
ULIndexSchema class [UltraLite.NET API]
ColumnCount property, 293
description, 290
GetColumnName method, 291
IsColumnDescending method, 292
IsForeignKey property, 293
IsForeignKeyCheckOnCommit property, 293
IsForeignKeyNullable property, 294
IsOpen property, 294
IsPrimaryKey property, 295
IsUniqueIndex property, 295
IsUniqueKey property, 295
Name property, 296
ReferencedIndexName property, 296
ReferencedTableName property, 297
ULInfoMessageEventArgs class [UltraLite.NET API]
description, 297
Message property, 298
NativeError property, 299
Source property, 299
ToString method, 298
ULInfoMessageEventHandler delegate
[UltraLite.NET API]
description, 437
ULMetaDataCollectionNames class [UltraLite.NET API]
Columns property, 300
DataSourceInformation property, 301
DataTypes property, 301
description, 299
ForeignKeys property, 302
IndexColumns property, 303
Indexes property, 303
MetaDataCollections property, 304
Publications property, 305
ReservedWords property, 305
Restrictions property, 306
Tables property, 306
ULParameter class [UltraLite.NET API]
DbType property, 315
description, 307
Direction property, 316
IsNullable property, 317
Offset property, 317
ParameterName property, 317
Precision property, 318
ResetDbType method, 315
Scale property, 318
Size property, 319
SourceColumn property, 319
SourceColumnNullMapping property, 320
SourceVersion property, 320
ToString method, 315
ULDbType property, 320
ULParameter constructor, 309
Value property, 321
ULParameter constructor
ULParameter class [UltraLite.NET API], 309
ULParameterCollection class [UltraLite.NET API]
Add method, 323
AddRange method, 329
Clear method, 330
Contains method, 331
CopyTo method, 332
Count property, 336
description, 321
GetEnumerator method, 332
IndexOf method, 333
Insert method, 334
IsFixedSize property, 336
IsReadOnly property, 337
IsSynchronized property, 337
Remove method, 334
RemoveAt method, 335
SyncRoot property, 337
this property, 338
ULResultSet class [UltraLite.NET API]
AppendBytes method, 345
AppendChars method, 347
Delete method, 348
description, 339
SetBoolean method, 348
SetByte method, 349
SetBytes method, 350
SetDateTime method, 351
SetDBNull method, 351
SetDecimal method, 352
SetDouble method, 353
SetFloat method, 353

SetGuid method, 354
SetInt16 method, 355
SetInt32 method, 356
SetInt64 method, 356
SetString method, 357
SetTimeSpan method, 358
SetToDefault method, 359
SetUInt16 method, 359
SetUInt32 method, 360
SetUInt64 method, 361
Update method, 362
UpdateBegin method, 362
ULResultSetSchema class [UltraLite.NET API]
  description, 362
  Name property, 364
ULRowsCopied event
ULBulkCopy class [UltraLite.NET API], 56
ULRowsCopiedEventArgs class [UltraLite.NET API]
  Abort property, 365
  description, 364
  RowsCopied property, 365
  ULRowsCopiedEventArgs constructor, 365
ULRowsCopiedEventArgs constructor
ULRowsCopiedEventArgs class [UltraLite.NET API], 365
ULRowsCopiedEventHandler delegate
[UltraLite.NET API]
  description, 439
ULRowUpdatedEventArgs class [UltraLite.NET API]
  Command property, 368
  description, 366
  RecordsAffected property, 368
  ULRowUpdatedEventArgs constructor, 367
ULRowUpdatedEventArgs constructor
ULRowUpdatedEventArgs class [UltraLite.NET API], 367
ULRowUpdatedEventHandler delegate
[UltraLite.NET API]
  description, 438
ULRowUpdatingEventArgs class [UltraLite.NET API]
  Command property, 371
  description, 369
  ULRowUpdatingEventArgs constructor, 370
ULRowUpdatingEventArgs constructor
ULRowUpdatingEventArgs class [UltraLite.NET API], 370
ULRowUpdatingEventHandler delegate
[UltraLite.NET API]
  description, 438
ULRuntimeType enumeration [UltraLite.NET API]
  description, 446
ULServerSyncListener interface [UltraLite.NET API]
  description, 371
  ServerSyncInvoked method, 372
ULSqlProgressData class [UltraLite.NET API]
  CurrentScript property, 374
  description, 373
  ScriptCount property, 374
  State property, 374
ULSqlProgressState enumeration [UltraLite.NET API]
  description, 447
ULSyncType enumeration [UltraLite.NET API]
  description, 448
ULSyncParms class [UltraLite.NET API]
  AdditionalParms property, 377
  AuthenticationParms property, 378
  CopyFrom method, 377
  description, 375
  DownloadOnly property, 379
  KeepPartialDownload property, 379
  NewPassword property, 380
  Password property, 381
  PingOnly property, 381
  Publications property, 382
  ResumePartialDownload property, 382
  SendDownloadAck property, 383
  Stream property, 384
  StreamParms property, 384
  ToString method, 377
  UploadOnly property, 385
  UserName property, 385
  Version property, 386
ULSyncProgressData class [UltraLite.NET API]
  CurrentDownloadRowCount property, 388
  description, 386
  FLAG_IS_BLOCKING field, 395
  Flags property, 388
  IgnoredDeletes property, 389
  IgnoredUpdates property, 389
  IsFinalSyncProgress property, 389
  ReceivedBytes property, 390
  ReceivedDeletes property, 390
  ReceivedInserts property, 390
| **UltraLite.NET development**, 16 |
| **ULTableSchema class [UltraLite.NET API]** description, 421 |
| **GetColumnDefaultValue method**, 423 |
| **GetColumnPartitionSize method**, 424 |
| **GetIndex method**, 425 |
| **GetIndexName method**, 425 |
| **GetOptimalIndex method**, 426 |
| **GetPublicationPredicate method**, 426 |
| **IndexCount property**, 432 |
| **IsColumnAutoIncrement method**, 427 |
| **IsColumnCurrentDate method**, 428 |
| **IsColumnCurrentTime method**, 428 |
| **IsColumnCurrentTimestamp method**, 429 |
| **IsColumnGlobalAutoIncrement method**, 430 |
| **IsColumnNewUUID method**, 430 |
| **IsColumnNullable method**, 431 |
| **IsInPublication method**, 432 |
| **IsNeverSynchronized property**, 432 |
| **Name property**, 433 |
| **PrimaryKey property**, 433 |
| **UploadUnchangedRows property**, 434 |

| **UltraLite Visual Studio integration**, 25 |
| **UltraLite databases connecting in UltraLite.NET**, 3 |
| **UltraLite.NET API information access**, 16 |
| **UltraLite modes** |
| **UltraLite.NET about**, 3 |
| **accessing schema information**, 16 |
| **ActiveSync synchronization**, 19 |
| **data modification**, 5 |
| **data modification with SQL**, 5 |
| **data retrieval**, 8 |
| **deployment**, 19, 20, 21 |
| **dynamic SQL tutorial**, 25 |
| **supported platforms**, 1 |
| **synchronization example**, 18 |
| **synchronizing data**, 18 |
| **Table API introduction**, 9 |
| **transaction processing**, 15 |
| **tutorials**, 25 |

| **UltraLite.NET API architecture**, 43 |
| **ULActiveSyncListener interface**, 44 |

---

| **ReceivedUpdates property**, 391 |
| **SentBytes property**, 391 |
| **SentDeletes property**, 392 |
| **SentInserts property**, 392 |
| **SentUpdates property**, 392 |
| **State property**, 393 |
| **SyncTableCount property**, 393 |
| **SyncTableIndex property**, 393 |
| **TableID property**, 394 |
| **TableName property**, 394 |
| **TotalDownloadRowCount property**, 394 |
| **TruncateDeletes property**, 395 |

| **ULSyncProgressedDlg delegate [UltraLite.NET API]** description, 439 |
| **ULSyncProgressListener interface [UltraLite.NET API]** description, 396 |
| **SyncProgressed method**, 396 |
| **ULSyncProgressState enumeration [UltraLite.NET API]** description, 449 |
| **ULTResult class [UltraLite.NET API]** |
| **AuthStatus property**, 398 |
| **AuthValue property**, 398 |
| **description**, 397 |
| **IgnoredRows property**, 398 |
| **PartialDownloadRetained property**, 399 |
| **StreamErrorCode property**, 399 |
| **StreamErrorParameters property**, 399 |
| **StreamErrorSystem property**, 400 |
| **Timestamp property**, 400 |
| **UploadOK property**, 400 |

| **ULTTable class [UltraLite.NET API]** |
| **DeleteAllRows method**, 407 |
| **description**, 401 |
| **FindBegin method**, 407 |
| **FindFirst method**, 408 |
| **FindLast method**, 410 |
| **FindNext method**, 412 |
| **FindPrevious method**, 414 |
| **Insert method**, 415 |
| **InsertBegin method**, 416 |
| **LookupBackward method**, 416 |
| **LookupBegin method**, 418 |
| **LookupForward method**, 419 |
| **Schema property**, 421 |
| **Truncate method**, 420 |

| **ULTableSchema class** | **Index** | **468 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0** |
ULAuthStatusCode enumeration, 440
ULBulkCopy class, 46
ULBulkCopyColumnMapping class, 56
ULBulkCopyColumnMappingCollection class, 62
ULBulkCopyOptions enumeration, 441
ULCommand class, 71
ULCommandBuilder class, 107
ULConnection class, 118
ULConnectionParms class, 163
ULConnectionStringBuilder class, 172
ULCreateParms class, 188
ULCursorSchema class, 197
ULDataAdapter class, 204
ULDatabaseManager class, 214
ULDatabaseSchema class, 221
ULDataReader class, 228
ULDateOrder enumeration, 442
ULDbType enumeration, 442
ULDBValid enumeration, 441
ULException class, 265
ULFactory class, 267
ULFileTransfer class, 272
ULFileTransferProgressData class, 287
ULFileTransferProgressListener interface, 289
ULIndexSchema class, 290
ULInfoMessageEventArgs class, 297
ULInfoMessageEventHandler delegate, 437
ULMetaDataCollectionNames class, 299
ULParameter class, 307
ULParameterCollection class, 321
ULResultSet class, 339
ULResultSetSchema class, 362
ULRowsCopiedEventArgs class, 364
ULRowsCopiedEventHandler delegate, 439
ULRowUpdatedEventArgs class, 366
ULRowUpdatedEventHandler delegate, 438
ULRowUpdatingEventArgs class, 369
ULRowUpdatingEventHandler delegate, 438
ULRuntimeType enumeration, 446
ULServerSyncListener interface, 371
ULSqlProgressData class, 373
ULSqlProgressState enumeration, 447
ULStreamType enumeration, 448
ULSyncParms class, 375
ULSyncProgressData class, 386
ULSyncProgressedDlg delegate, 439
ULSyncProgressListener interface, 396
ULSyncProgressState enumeration, 449
ULSyncResult class, 397
ULTable class, 401
ULTableSchema class, 421
ULTransaction class, 434
UltraLite.NET tutorial
code listing for C# tutorial, 38
code listing for Visual Basic tutorial, 40
ULTransaction class [UltraLite.NET API]
Commit method, 435
Connection property, 436
description, 434
IsolationLevel property, 437
Rollback method, 436
Update method
ULResultSet class [UltraLite.NET API], 362
update mode
UltraLite.NET, 11
UpdateBegin method
ULResultSet class [UltraLite.NET API], 362
UpdateCommand property
ULDataAdapter class [UltraLite.NET API], 213
UpdatedRowSource property
ULCommand class [UltraLite.NET API], 106
updating
UltraLite.NET table rows, 12
UploadFile method
ULFileTransfer class [UltraLite.NET API], 277
UploadOK property
ULSyncResult class [UltraLite.NET API], 400
UploadOnly property
ULSyncParms class [UltraLite.NET API], 385
UploadUnchangedRows property
ULTableSchema class [UltraLite.NET API], 434
UserID property
ULConnectionParms class [UltraLite.NET API], 172
ULConnectionStringBuilder class [UltraLite.NET API], 187
UserName property
ULFileTransfer class [UltraLite.NET API], 286
ULSyncParms class [UltraLite.NET API], 385
UTF8Encoding property
ULCreateParms class [UltraLite.NET API], 196
V
ValidateDatabase method
ULConnection class [UltraLite.NET API], 153
ULDatabaseManager class [UltraLite.NET API], 220

Value property
ULParameter class [UltraLite.NET API], 321
values
UltraLite.NET accessing in, 14

Version property
ULFileTransfer class [UltraLite.NET API], 286
ULSyncParms class [UltraLite.NET API], 386

Visual Studio
integration with UltraLite, 25
UltraLite.NET connecting to UltraLite databases, 3

W

Windows Mobile
connecting to a database, 3
creating a database, 3
developing applications, 3
UltraLite.NET target platforms, 1

WriteToServer method
ULBulkCopy class [UltraLite.NET API], 51