UltraLite.NET™ User's Guide

Part number: DC50043-01-0902-01
Last modified: October 2004

Contents

About This Manual

Subject	This manual describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.
Audience	This manual is intended for .NET application developers who wish to take advantage of the performance, resource efficiency, robustness, and security of an UltraLite relational database for data storage and synchronization.

SQL Anywhere Studio documentation

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

The SQL Anywhere Studio documentation

The SQL Anywhere Studio documentation is available in a variety of forms: in an online form that combines all books in one large help file; as separate PDF files for each book; and as printed books that you can purchase. The documentation consists of the following books:

- **Introducing SQL Anywhere Studio** This book provides an overview of the SQL Anywhere Studio database management and synchronization technologies. It includes tutorials to introduce you to each of the pieces that make up SQL Anywhere Studio.
- **What's New in SQL Anywhere Studio** This book is for users of previous versions of the software. It lists new features in this and previous releases of the product and describes upgrade procedures.
- **Adaptive Server Anywhere Database Administration Guide** This book covers material related to running, managing, and configuring databases and database servers.
- **Adaptive Server Anywhere SQL User's Guide** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- **Adaptive Server Anywhere SQL Reference Manual** This book provides a complete reference for the SQL language used by Adaptive Server Anywhere. It also describes the Adaptive Server Anywhere system tables and procedures.
- **Adaptive Server Anywhere Programming Guide** This book describes how to build and deploy database applications using the C, C++, and Java programming languages. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by those tools. It also describes the Adaptive Server Anywhere ADO.NET data provider.
- **Adaptive Server Anywhere SNMP Extension Agent User's Guide** This book describes how to configure the Adaptive Server Anywhere SNMP Extension Agent for use with SNMP management applications to manage Adaptive Server Anywhere databases.
- **Adaptive Server Anywhere Error Messages** This book provides a complete listing of Adaptive Server Anywhere error messages together with diagnostic information.

- **SQL Anywhere Studio Security Guide** This book provides information about security features in Adaptive Server Anywhere databases. Adaptive Server Anywhere 7.0 was awarded a TCSEC (Trusted Computer System Evaluation Criteria) C2 security rating from the U.S. Government. This book may be of interest to those who wish to run the current version of Adaptive Server Anywhere in a manner equivalent to the C2-certified environment.
- **MobiLink Administration Guide** This book describes how to use the MobiLink data synchronization system for mobile computing, which enables sharing of data between a single Oracle, Sybase, Microsoft or IBM database and many Adaptive Server Anywhere or UltraLite databases.
- **MobiLink Clients** This book describes how to set up and synchronize Adaptive Server Anywhere and UltraLite remote databases.
- **MobiLink Server-Initiated Synchronization User's Guide** This book describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization from the consolidated database.
- **MobiLink Tutorials** This book provides several tutorials that walk you through how to set up and run MobiLink applications.
- **QAnywhere User's Guide** This manual describes MobiLink QAnywhere, a messaging platform that enables the development and deployment of messaging applications for mobile and wireless clients, as well as traditional desktop and laptop clients.
- **iAnywhere Solutions ODBC Drivers** This book describes how to set up ODBC drivers to access consolidated databases other than Adaptive Server Anywhere from the MobiLink synchronization server and from Adaptive Server Anywhere remote data access.
- **SQL Remote User's Guide** This book describes all aspects of the SQL Remote data replication system for mobile computing, which enables sharing of data between a single Adaptive Server Anywhere or Adaptive Server Enterprise database and many Adaptive Server Anywhere databases using an indirect link such as e-mail or file transfer.
- **SQL Anywhere Studio Help** This book includes the context-sensitive help for Sybase Central, Interactive SQL, and other graphical tools. It is not included in the printed documentation set.
- **UltraLite Database User's Guide** This book is intended for all UltraLite developers. It introduces the UltraLite database system and provides information common to all UltraLite programming interfaces.

- **UltraLite Interface Guides** A separate book is provided for each UltraLite programming interface. Some of these interfaces are provided as UltraLite components for rapid application development, and others are provided as static interfaces for C, C++, and Java development.

In addition to this documentation set, PowerDesigner and InfoMaker include their own online documentation.

Documentation formats

SQL Anywhere Studio provides documentation in the following formats:

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

  To access the online documentation on Windows operating systems, choose Start ➤ Programs ➤ SQL Anywhere 9 ➤ Online Books. You can navigate the online documentation using the HTML Help table of contents, index, and search facility in the left pane, as well as using the links and menus in the right pane.

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

- **PDF books** The SQL Anywhere books are provided as a set of PDF files, viewable with Adobe Acrobat Reader.

  The PDF books are accessible from the online books, or from the Windows Start menu.

- **Printed books** The complete set of books is available from Sybase sales or from eShop, the Sybase online store, at *http://eshop.sybase.com/eshop/documentation*.

Documentation conventions

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

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

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

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

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

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

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

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

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

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

  RELEASE SAVEPOINT [*savepoint-name*]

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

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

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

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

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

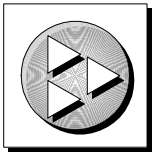
  [**QUOTES** { **ON** | **OFF** }]

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

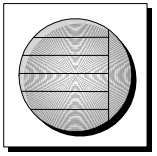
Graphic icons

The following icons are used in this documentation.

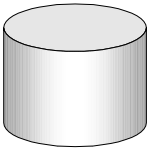
- A client application.

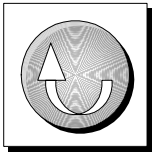
- A database server, such as Sybase Adaptive Server Anywhere.

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

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

- A programming interface.

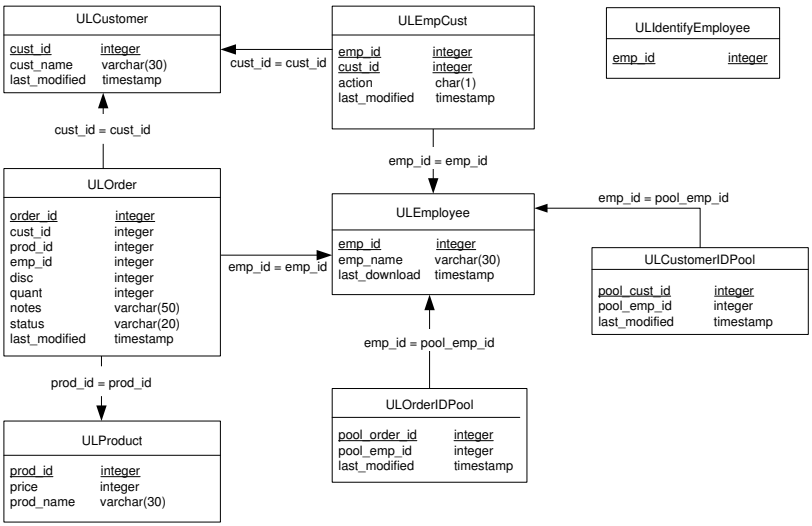
The CustDB sample database

Many of the examples in the MobiLink and UltraLite documentation use the UltraLite sample database.

The reference database for the UltraLite sample database is held in a file named *custdb.db*, and is located in the *Samples\UltraLite\CustDB* subdirectory of your SQL Anywhere directory. A complete application built on this database is also supplied.

The sample database is a sales-status database for a hardware supplier. It holds customer, product, and sales force information for the supplier.

The following diagram shows the tables in the CustDB database and how they are related to each other.



Finding out more and providing feedback

Finding out more

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

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

When you write to one of these newsgroups, always provide detailed information about your problem, including the build number of your version of SQL Anywhere Studio. You can find this information by typing **dbeng9 -v** at a command prompt.

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

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

Newsgroup disclaimer

iAnywhere Solutions has no obligation to provide solutions, information or ideas on its newsgroups, nor is iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and ensure its operation and availability.

iAnywhere Solutions Technical Advisors as well as other staff assist on the newsgroup service when they have time available. They offer their help on a volunteer basis and may not be available on a regular basis to provide solutions and information. Their ability to help is based on their workload.

Feedback

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

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

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

CHAPTER 1

Introduction to UltraLite.NET

About this chapter

This chapter introduces you to UltraLite.NET. It assumes that you are familiar with the features of UltraLite, as described in "Welcome to UltraLite" [*UltraLite Database User's Guide*, page 3].

☞ For more information about creating applications using UltraLite.NET, see "Understanding UltraLite.NET Development" on page 21.

☞ For a hands-on tutorial introducing UltraLite.NET, see "Tutorial: Build an UltraLite.NET Application" on page 5.

Contents

UltraLite and .NET

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

UltraLite.NET provides the following namespaces:

- **iAnywhere.Data.UltraLite** This namespace provides an ADO.NET interface to UltraLite. It has the virtues of being built on an industry-standard model and of providing a migration path to the Adaptive Server Anywhere ADO.NET interface, which is very similar.
- **iAnywhere.UltraLite** This namespace is the original UltraLite.NET namespace. It is deprecated in release 9.0.2, and will not be available in the next major release of SQL Anywhere.

The UltraLite.NET runtime supports both these namespaces.

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

System requirements and supported platforms

Development platforms

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

- Microsoft Windows NT/2000/XP.
- Visual Studio.NET or Visual Studio.NET 2003.
- For Windows CE devices, .NET Compact Framework version 1.0.5000 or later.

Target platforms

UltraLite.NET supports the following target platforms:

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

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

- **iAnywhere.Data.UltraLite.dll** An assembly containing the iAnywhere.Data.UltraLite ADO.NET namespace. This file is required only if your application uses the iAnywhere.Data.UltraLite namespace.
- **iAnywhere.Data.UltraLite.resources.dll** The resources needed by the iAnywhere.Data.UltraLite ADO.NET namespace. This file is required only if your application uses the iAnywhere.Data.UltraLite namespace.
- **iAnywhere.UltraLite.dll** An assembly containing the iAnywhere.UltraLite namespace. This file is required only if your application uses the iAnywhere.UltraLite namespace.
- **iAnywhere.UltraLite.resources.dll** The resources needed by the iAnywhere.UltraLite namespace. This file is required only if your application uses the iAnywhere.UltraLite namespace.
- **ulnet9.dll** This file contains the UltraLite runtime. A separate version of the runtime is provided for each target platform.
- **UltraLite schema file** The UltraLite runtime library uses the information in the schema file to set the database schema. Once a database file is created, the schema file is no longer required.

☞ For more information, see "UltraLite development platforms" [*Introducing SQL Anywhere Studio*, page 99] and "UltraLite target platforms" [*Introducing SQL Anywhere Studio*, page 109].

UltraLite.NET architecture

The UltraLite.NET namespaces are named iAnywhere.Data.UltraLite (ADO.NET interface) and iAnywhere.UltraLite. It is recommended that new applications use the iAnywhere.Data.UltraLite namespace.

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

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

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

- **DatabaseManager** You create one DatabaseManager object for each application.
- **Connection** Each Connection object represents a connection to an UltraLite database. You can create one or more Connection objects.
- **Table** The Table object provides access to the data in the database.
- **PreparedStatement, ResultSet, and ResultSetSchema** These dynamic SQL objects allow you to create Dynamic SQL statements, make queries and execute INSERT, UPDATE and DELETE statements, and attain programmatic control over database result sets.
- **SyncParms** You use the SyncParms object to synchronize your UltraLite database with a MobiLink synchronization server.

☞ For more information, see "iAnywhere.Data.UltraLite namespace" on page 49 for the ADO.NET interface and "iAnywhere.UltraLite namespace" on page 339 for the component interface.

CHAPTER 2

Tutorial: Build an UltraLite.NET Application

About this chapter

This chapter provides a tutorial to guide you through the process of building an UltraLite.NET application in Visual Studio .NET 2003.

Contents

Introduction

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

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

Timing

The tutorial takes about thirty minutes if you cut and paste the code. It takes significantly longer if you type the code yourself.

Competencies and experience

This tutorial assumes the following:

- You are familiar with the C# programming language or the Visual Basic .NET programming language.
- You have Microsoft Visual Studio .NET 2003 installed on your computer.
- You know how to create an UltraLite schema using the UltraLite Schema Painter.

  ☞ For more information, see "Lesson 1: Create an UltraLite database schema" [*UltraLite Database User's Guide*, page 130].

Goals

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

Lesson 1: Create a Visual Studio project

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

❖ To create a Visual Studio project

1. Create a Visual Studio project .
   - From the Visual Studio .NET 2003 File menu, choose New ➤ Project to create a new project.
     The New Project window appears.
   - In the left pane, select either the Visual Basic Projects folder or the Visual C# Projects folder.
   - In the right pane, select a Smart Device Application and name your project **VBApp** or **CSApp**, depending on whether you choose Visual Basic or C# for the programming language.
   - Enter a Location of *c:\tutorial\uldotnet* and click OK.
     The Smart Device Application Wizard appears.
   - Choose Pocket PC as the target platform, and select Windows Application as the project type. Click OK.
     A Design workspace appears, displaying a form.
2. Add references to your project.
   - Add the iAnywhere.Data.UltraLite assembly and the associated resources to your project.
     a. From the Project menu, choose Add Reference.
     The Add Reference window appears.
     b. Select **iAnywhere.Data.UltraLite (CE)** from the list of available references. Click Select to add it to the list of selected components. If this reference does not appear in the list, click Browse and locate it in the *ultralite\UltraLite.NET\ce* subdirectory of your SQL Anywhere installation. Select *iAnywhere.Data.UltraLite.dll* and click Open.
     c. Select **iAnywhere.Data.UltraLite (CE) EN** from the list of available references. Click Select to add it to the list of selected components.
     If this reference does not appear in the list, click Browse and locate it in the *ultralite\UltraLite.NET\ce\xx* subdirectory of your SQL Anywhere installation, where xx is a two-letter abbreviation for the language. Select *iAnywhere.Data.UltraLite.resources.dll* and click Open.

d. Click OK to add the assembly and resources to your project.

- Link the UltraLite component to your project.

  In this step, ensure that you add a link to the component, and that you do not Open the component.

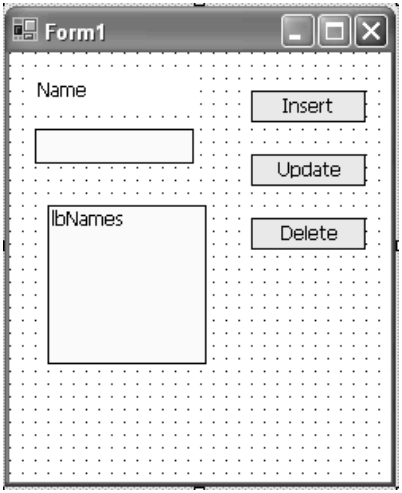
  a. From the Project menu, choose Add Existing Item and browse to the *ultralite\UltraLite.NET\ce* subdirectory of your SQL Anywhere installation.
  b. Set the Files of Type drop down to All Files so that DLL files are displayed.
  c. Open the folder corresponding to the processor of the CE device you are using (for the Pocket PC emulator, open the x86 folder) and select *ulnet9.dll. Do not click Open.*
  d. Click the arrow on the Open button and select **Link File** to link it to your project.

3. Create a form for your application.

   Add the following visual components to the form.

Type	Name	Text
Button	btnInsert	Insert
Button	btnUpdate	Update
Button	btnDelete	Delete
TextBox	txtName	
ListBox	lbNames	
Label	laName	Name

   Your form should look like the following figure.



4. Build and deploy your solution.

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

   a. From the Build menu, choose Build Solution. Confirm that the project builds successfully.

   b. From the Debug menu, choose Start. This action deploys your application to the device or emulator, and starts it. The application is deployed to *\Program Files\VBApp* or *\Program Files\CSApp* depending on your project name.

      The deployment may take some time.

   c. Confirm that the application deploys to the emulator or your target device.

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

Lesson 2: Create an UltraLite schema file

The following procedure creates an UltraLite database schema using the UltraLite Schema Painter.

☞ For information on using the Schema Painter, see "Lesson 1: Create an UltraLite database schema" [*UltraLite Database User's Guide*, page 130].

❖ To create a schema file

1. Start the UltraLite Schema Painter:

   From the Start menu, choose Programs ➤ SQL Anywhere 9 ➤ UltraLite ➤ UltraLite Schema Painter.

2. Use the UltraLite Schema Painter to create a database schema in the same directory as your application with the following characteristics.
   - **Schema file name** *vbapp.usm* or *csapp.usm*, depending on your application.
   - **Collation sequence** Default
   - **Case sensitivity** Leave unchecked
   - **Table name** Names
   - **Columns** Create columns in the Names table with the following attributes:

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

   - **Primary key** Ascending ID

3. Link the schema file to your application.
   - From the Visual Studio .NET 2003 interface, choose Project ➤ Add Existing Item.
   - Browse to your tutorial directory and select *csapp.usm* or *vbapp.usm*, as appropriate. *Do not click Open.*
   - Click the arrow on the Open button and choose Link File.
   - In the Solution Explorer, right click your schema file and select Properties. In the properties pane, set the Build Action property to **Content**.

Lesson 3: Connect to the database

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

If an UltraLite application attempts to connect to a database that does not exist, it uses the schema file to create the database.

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

❖ To add an UltraLite connection to your application

1. Add the ConnectionParms control to your form.

   From the UltraLite.NET CF tab in the toolbox, choose ULConnectionParms and add it to your form. The control is named UlConnectionParms1 (Visual Basic) or ulConnectionParms1 (C#).

   If you have not rebooted since installing SQL Anywhere Studio, you may get an error message at this point. Reboot your machine to solve this problem.

2. Specify the connection parameters.

   Ensure the following properties are set:

   - **DatabaseOnCE** *\Program Files\CSApp\CSApp.udb* for a Visual C# application, or *\Program Files\VBApp\VBApp.udb* for a Visual Basic application.
   - **SchemaOnCE** *\Program Files\CSApp\CSApp.usm* for a Visual C# application, or *\Program Files\VBApp\VBApp.usm* for a Visual Basic application.

3. Open the source code for the form.

   Double-click the form to open the source file.

4. 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
   ```

5. Add global variables to the form declaration.

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

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

   For a Visual Basic project, add the following code after the declaration of the form components (FriendsWithEvents declarations) and before the first method declaration.

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

   These variables are used as follows:

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

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

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

   This method performs the following tasks.

   - Opens a connection to the database using the connection parameters set in the ULConnectionParms1 control.
   - Calls the RefreshListBox method, which you will define later in this tutorial.
   - If an error occurs, prints the error message. For SQL errors, the code also prints the error code. For more information about the error code, see *ASA Error Messages*.

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

   ```
   //Visual C#
   try {
       Conn = new ULConnection(
           ulConnectionParms1.ToString() );
       Conn.OpenWithCreate();
       Conn.DatabaseID = 1;
       RefreshListBox();
   }
   catch ( System.Exception t ) {
       MessageBox.Show( "Exception: " + t.Message);
   }
   ```

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

```
'Visual Basic
Try
    Conn = New ULConnection( _
        UlConnectionParms1.ToString())
    Conn.OpenWithCreate()
    Conn.DatabaseID = 1
    RefreshListBox()
Catch
    MsgBox("Exception: " + err.Description)
End Try
```

7. Build the project.

   From the Build menu choose Build Solution. At this stage, you should get a single error reported, which is that the Name RefreshListBox is not declared. You will fix this error by adding the function in the next lesson.

   If you get other errors, fix them. Check for common errors, such as case inconsistencies in C#. For example, **UltraLite** and **ULConnection** must match case exactly.

Lesson 4: Insert, update, and delete data

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

☞ For more information, see "Choosing between components and static interfaces" [*UltraLite Database User's Guide*, page 11] or "Accessing and manipulating data with the Table API" on page 36.

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

❖ To add code to maintain the listbox

1. Right-click the form and select 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 the Names table in the database.
   - Executes the query, returning a result set as a ULDataReader.
   - Instantiates an integer array with length equal to the number of rows in the result set.
   - Populates the listbox with the names returned in the ULDataReader and populates the integer array with the ids returned in the ULDataReader.
   - Closes the ULDataReader.
   - If an error occurs, prints the error message. For SQL errors, the code also prints the error code. For more information about the error code, see *ASA Error Messages*.

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

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

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

```
'Visual Basic
Private Sub RefreshListBox()
    Try
        Dim cmd As ULCommand = Conn.CreateCommand()
        Dim i As Integer = 0
        lbNames.Items.Clear()
        cmd.CommandText = "SELECT ID, Name FROM Names"
        Dim dr As ULDataReader = cmd.ExecuteReader()
        ReDim ids(dr.RowCount)
        While (dr.MoveNext)
            lbNames.Items.Add(dr.GetString(1))
            ids(i) = dr.GetInt32(0)
            i = i + 1
        End While
        dr.Close()
    Catch ex As Exception
        MsgBox(ex.ToString)
    End Try
End Sub
```

3. Build the project.

   Building the project should result in no errors.

❖ To implement INSERT, UPDATE and DELETE

1. Double-click the Insert button to create a btnInsert_Click method. This method carries out the following tasks.
   - ♦ Instantiates a ULCommand object and assigns it an INSERT statement that inserts the value in the textbox into the database.
   - ♦ Executes the statement.
   - ♦ Disposes of the ULCommand object.
   - ♦ Refreshes the listbox.
   - ♦ If an error occurs, prints the error message. For SQL errors, the code also prints the error code. For more information about the error code, see *ASA Error Messages*.

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

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

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

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

2. Double-click the Update button to create a btnUpdate_Click method. This method carries out the following tasks.
   - ♦ Instantiates a ULCommand object and assigns it an UPDATE statement that inserts the value in the textbox into the database based on the associated id.

- Executes the statement.
- Disposes of the ULCommand object.
- Refreshes the listbox.
- If an error occurs, prints the error message. For SQL errors, the code also prints the error code. For more information about the error code, see *ASA Error Messages*.

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

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

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

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

3. Double-click the Delete button to create a btnDelete_Click method. Add code to carry out the following tasks.
   - Instantiates a ULCommand object and assigns it a DELETE statement. The DELETE statement deletes the selected row from the database, based on the associated id from the integer array ids.
   - Executes the statement.

- Disposes of the ULCommand object.
- Refreshes the listbox.
- If an error occurs, prints the error message. For SQL errors, the code also prints the error code. For more information about the error code, see *ASA Error Messages*.

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

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

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

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

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

Lesson 5: Build and deploy your application

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

❖ To deploy your application

1. Build the solution.

   Ensure that your application builds without errors.

2. Select Debug ➤ Start.

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

Deployment troubleshooting checklist

If errors are reported, you may wish 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 schema file is located in the same directory as the application.
- Confirm that the path to the schema file in your application code is correct. For more information, see "Lesson 3: Connect to the database" on page 11.
- Confirm that you chose Link File when adding the schema file to the project and that you set the Build Action to Content. If you did not, the files will not be deployed to the device.
- Ensure that you added a reference to the correct version of *ulnet9.dll* for your target platform, or ran the Windows CE installer. If you switch between the emulator and a real device, you must change the version of this library that you use. For more information, see "Lesson 1: Create a Visual Studio project" on page 7.
- You may wish to exit the emulator, without saving state. Redeploying copies all required files to the emulator, and ensures there are no version problems.

❖ To test your application

1. Insert data into the database.

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

2. Update data in the database.

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

3. Delete data from the database.

   Select a name from the listbox. Click Delete. The name should no longer appear in the listbox.

This completes the tutorial.

CHAPTER 3

Understanding UltraLite.NET Development

About this chapter

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

☞ For hands-on tutorials, see "Tutorial: Build an UltraLite.NET Application" on page 5.

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

Contents

UltraLite database schemas

The database schema is a description of the database. It is the collection of tables, indexes, keys, and publications within the database, and all the relationships between them.

You do not alter the schema of an UltraLite database directly. Instead, you create a schema (*.usm*) file and upgrade the database schema from that file by calling a built-in UltraLite function in your application.

A schema file is also used in the initial creation of a database to specify the structure of the database.

Creating UltraLite database schema files

You can create an UltraLite schema file using the UltraLite Schema Painter or the *ulinit* utility.

- **UltraLite Schema Painter** The UltraLite Schema Painter is a graphical utility for creating and editing UltraLite schema files.

  To start the Schema painter, choose Start ➤ Programs ➤ SQL Anywhere 9 ➤ UltraLite ➤ UltraLite Schema Painter, or double-click a schema (*.usm*) file in Windows Explorer.

  ☞ For more information about using the UltraLite Schema Painter, see "Lesson 1: Create an UltraLite database schema" [*UltraLite Database User's Guide*, page 130].

- **The ulinit utility** If you have the Adaptive Server Anywhere database management system, you can generate an UltraLite schema file using the *ulinit* command line utility.

  ☞ For more information about using the *ulinit* utility, see "The ulinit utility" [*UltraLite Database User's Guide*, page 112].

Upgrading your database schema

To modify your existing database structure, use the ULDatabaseSchema.ApplyFile method. In most cases there will be no data loss, however, data loss can occur if columns are deleted or if the data type for a column is changed to an incompatible type.

☞ For more information, see "ULDatabaseSchema members" on page 142 (iAnywhere.Data.UltraLite namespace) or "DatabaseSchema members" on page 413 (iAnywhere.UltraLite namespace).

Example

The following code applies a new schema file using the ApplyFile method. It assumes that you have already connected to the database as shown in

“Connecting to a database” on page 24.

```
// iAnywhere.Data.UltraLite namespace
ULDatabaseSchema schema = conn.Schema;
ULConnectionParms parms = new ULConnectionParms();
parms.SchemaOnDesktop = @"c:\tutorial\tutcustomer.usm";
try {
  schema.ApplyFile(parms );
} catch {
  // handle error
}
```

```
// iAnywhere.UltraLite namespace
DatabaseSchema schema = conn.Schema;
SchemaParms parms = new SchemaParms();
parms.SchemaOnDesktop = @"c:\tutorial\tutcustomer.usm";
try {
  schema.ApplyFile( parms );
} catch {
  // handle error
}
```

Connecting to a database

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

Using the Connection object

The following properties of the Connection 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 build transactions into your application.

  ☞ For more information, see "Transaction processing in UltraLite" on page 43.

- **User authentication** You can change the user ID and password for the application from the default values of DBA and SQL by using methods to Grant and Revoke connection permissions. Each application can have a maximum of four user IDs.

  ☞ For more information, see "User authentication" on page 46.

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

  ☞ For more information, see "Synchronizing UltraLite applications" on page 47.

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

  ☞ For more information, see "Accessing and manipulating data with the Table API" on page 36.

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

  ☞ For more information, see "Accessing and manipulating data using dynamic SQL" on page 29.

☞ For more information, see "ULConnection class" on page 72 (iAnywhere.Data.UltraLite namespace) or "Connection class" on page 345 (iAnywhere.UltraLite namespace).

Multi-threaded applications

Each Connection 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.

❖ To connect to an UltraLite database

1. Create and initialize a DatabaseManager object.

   This step is not required if using the iAnywhere.Data.UltraLite namespace.

   The DatabaseManager object is at the root of the object hierarchy. You create only one DatabaseManager object per application. It is often best to declare the DatabaseManager object as global to the application.

   ```
   // iAnywhere.UltraLite namespace
   DatabaseManager dbMgr = new DatabaseManager();
   ```

   ☞ For more information, see "DatabaseManager class" on page 399 (iAnywhere.UltraLite namespace).

2. Declare a Connection 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 Connection object as global to the application.

   ```
   // iAnywhere.Data.UltraLite namespace
   ULConnection conn;
   ```

   ```
   // iAnywhere.UltraLite namespace
   Connection conn;
   ```

3. Open a connection to an existing database, or create a new database if the specified database file does not exist.

   Most UltraLite applications deploy a schema file rather than a database file, and let UltraLite create the database file on the first connection attempt. Thus, the following code attempts to connect to an existing database. If the database file does not exist, the application creates a database file.

   ```
   // iAnywhere.Data.UltraLite namespace
   ULConnectionParms parms = new ULConnectionParms();
   // specify the location of the database file
   parms.DatabaseOnDesktop = "mydata.udb";
   // specify the location of the schema file
   parms.SchemaOnDesktop = "mydata.usm";
   conn = new ULConnection( parms.ToString() );
   conn.OpenWithCreate();
   ```

```
// iAnywhere.UltraLite namespace
ConnectionParms parms = new ConnectionParms();
// specify the location of the database file
parms.DatabaseOnDesktop = "mydata.udb";
try {
  conn = dbMgr.OpenConnection( parms );
} catch( SQLException econn ) {
  if( econn.GetErrorCode() ==
    SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND ){
    CreateParms parms = new CreateParms();
    parms.DatabaseOnDesktop = "mydata.udb";
    parms.Schema.SchemaOnDesktop = "mydata.usm";
    try {
      conn = dbMgr.CreateDatabase( parms );
  }
}
```

Example

The following code opens a connection to an UltraLite database named *mydata.udb*. If the database does not exist, it is created using the UltraLite schema file named *mydata.usm*.

```
// iAnywhere.Data.UltraLite namespace
ULConnectionParms parms = new ULConnectionParms();
parms.DatabaseOnDesktop = "mydata.udb";
parms.SchemaOnDesktop = "mydata.usm";
conn = new ULConnection( parms.ToString() );
conn.InfoMessage += new ULInfoMessageEventHandler(
    InfoMessageEventHandler );
conn.OpenWithCreate();
```

```
// iAnywhere.UltraLite namespace
CreateParms parms = new CreateParms();
parms.DatabaseOnDesktop = "mydata.udb";
parms.Schema.SchemaOnDesktop = "mydata.usm";
try {
  conn = dbMgr.OpenConnection( parms );
  System.Console.WriteLine(
    "Connected to an existing database." );
}
catch( SQLException econn ) {
  if( econn.GetErrorCode() ==
      SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND ){
    conn = dbMgr.CreateDatabase( parms );
    System.Console.WriteLine(
      "Connected to a new database." );
  } else {
    throw econn;
  }
}
```

The above code uses the InfoMessage handler defined below to detect if the database is new.

```
// iAnywhere.Data.UltraLite namespace
protected void InfoMessageHandler(
    object obj, ULInfoMessageEventArgs args )
{
  if( args.NativeError == ULSQLCode.SQLE_DATABASE_CREATED ){
    System.Console.WriteLine( args.Message );
  }
}
```

Encryption and obfuscation

You can encrypt or obfuscate your UltraLite database using UltraLite.NET.

Encryption

To create a database with encryption, specify an encryption key using the ULConnectionParms.EncryptionKey property, then call the OpenWithCreate method. When you call the OpenWithCreate method, the database is created and encrypted with the specified key.

☞ For more information, see "Encryption Key connection parameter " [*UltraLite Database User's Guide*, page 75].

You can change the encryption key by specifying the new encryption key with the Connection.ChangeEncryptionKey method.

☞ For more information, see "ULConnection class" on page 72 and "ULConnectionParms class" on page 100 (iAnywhere.Data.UltraLite namespace) or "Connection class" on page 345 (iAnywhere.UltraLite namespace).

After the database is encrypted, connections to the database must specify the correct encryption key. Otherwise, the connection fails.

Obfuscation

To obfuscate the database, specify obfuscate=1 as a creation parameter.

☞ For more information about database encryption, see "Encrypting UltraLite databases" [*UltraLite Database User's Guide*, page 36].

Accessing and manipulating data using dynamic SQL

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

☞ For information about using the Table API, see "Accessing and manipulating data with the Table API" on page 36.

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

- Inserting, deleting, and updating rows.
- Retrieving rows to a result set.
- Scrolling through the rows of a result set.

☞ This section does not describe the SQL language itself. For information about dynamic SQL features, see "Dynamic SQL" [*UltraLite Database User's Guide*, page 159].

☞ For an overview of the sequence of operations required for any SQL operation, see "Using dynamic SQL" [*UltraLite Database User's Guide*, page 161].

Data manipulation: INSERT, UPDATE and DELETE

With UltraLite, you can perform SQL Data Manipulation Language operations. These operations are performed using the ULCommand.ExecuteNonQuery method (iAnywhere.Data.UltraLite namespace) or the PreparedStatement.executeStatement method (iAnywhere.UltraLite namespace).

☞ For more information, see "ULCommand class" on page 56 (iAnywhere.Data.UltraLite namespace) or "PreparedStatement class" on page 427 (iAnywhere.UltraLite namespace).

Using parameters with your iAnywhere.Data.UltraLite commands
Placeholders for parameters in SQL statements are indicated 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.

Using parameters in your iAnywhere.UltraLite prepared statements
Placeholders for parameters in SQL statements are indicated the ? character. For any INSERT, UPDATE or DELETE, each ? is referenced according to its ordinal position in the prepared statement. For example, the first ? is referred to as 1, and the second as 2.

❖ To INSERT a row

1. Declare a ULCommand.

```
// iAnywhere.Data.UltraLite namespace
ULCommand cmd;

// iAnywhere.UltraLite namespace
PreparedStatement prepStmt;
```

2. Assign a SQL statement to the ULCommand object.

```
// iAnywhere.Data.UltraLite namespace
cmd = conn.CreateCommand();
cmd.Command =
   "INSERT INTO MyTable(MyColumn) values (?)";

// iAnywhere.UltraLite namespace
prepStmt = conn.PrepareStatement(
   "INSERT INTO MyTable(MyColumn) values (?)");
```

3. Assign input parameter values for the statement.

   The following code shows a string parameter.

```
// iAnywhere.Data.UltraLite namespace
String newValue;
// assign value
cmd.add("", newValue);

// iAnywhere.UltraLite namespace
String newValue;
// assign value
prepStmt.SetStringParameter(1, newValue);
```

4. Execute the statement.

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

```
// iAnywhere.Data.UltraLite namespace
int rowsInserted = cmd.ExecuteNonQuery();

// iAnywhere.UltraLite namespace
long rowsInserted = prepStmt.ExecuteStatement();
```

5. If you are using transactions (AutoCommit false in the iAnywhere.UltraLite namespace), commit the change.

```
// iAnywhere.Data.UltraLite namespace
cmd.Transaction.Commit();

// iAnywhere.UltraLite namespace
conn.Commit();
```

❖ To UPDATE a row

1. Declare a ULCommand.

```
// iAnywhere.Data.UltraLite namespace
ULCommand cmd;

// iAnywhere.UltraLite namespace
PreparedStatement prepStmt;
```

2. Assign a statement to the ULCommand object.

```
// iAnywhere.Data.UltraLite namespace
cmd = conn.CreateCommand();
cmd.Command =
   "UPDATE MyTable SET MyColumn1 = ? WHERE MyColumn2 = ?";

// iAnywhere.UltraLite namespace
prepStmt = conn.PrepareStatement(
   "UPDATE MyTable SET MyColumn1 = ? WHERE MyColumn2 = ?");
```

3. Assign input parameter values for the statement.

```
// iAnywhere.Data.UltraLite namespace
String newValue;
String oldValue;
// assign values
cmd.add("", newValue);
cmd.add("", oldValue);

// iAnywhere.UltraLite namespace
String newValue;
String oldValue;
// assign values
prepStmt.SetStringParameter(1, newValue);
prepStmt.SetStringParameter(2, oldValue);
```

4. Execute the statement.

```
// iAnywhere.Data.UltraLite namespace
int rowsUpdated = cmd.ExecuteNonQuery();

// iAnywhere.UltraLite namespace
long rowsUpdated = prepStmt.ExecuteStatement();
```

5. If you are using transactions (AutoCommit false in the iAnywhere.UltraLite namespace), commit the change.

```
// iAnywhere.Data.UltraLite namespace
cmd.Transaction.Commit();
```

```
// iAnywhere.UltraLite namespace
conn.Commit();
```

❖ To DELETE a row

1. Declare a ULCommand.

```
// iAnywhere.Data.UltraLite namespace
ULCommand cmd;
```

```
// iAnywhere.UltraLite namespace
PreparedStatement prepStmt;
```

2. Assign a statement to the ULCommand object.

```
// iAnywhere.Data.UltraLite namespace
cmd = conn.CreateCommand();
cmd.Command =
   "DELETE FROM MyTable WHERE MyColumn = ?";
```

```
// iAnywhere.UltraLite namespace
prepStmt = conn.PrepareStatement(
   "DELETE FROM MyTable WHERE MyColumn = ?");
```

3. Assign input parameter values for the statement.

```
// iAnywhere.Data.UltraLite namespace
String deleteValue;
// assign value
cmd.add("", deleteValue);
```

```
// iAnywhere.UltraLite namespace
String deleteValue;
prepStmt.SetStringParameter(1, deleteValue);
```

4. Execute the statement.

```
// iAnywhere.Data.UltraLite namespace
int rowsDeleted = cmd.ExecuteNonQuery();
```

```
// iAnywhere.UltraLite namespace
long rowsDeleted = prepStmt.ExecuteStatement();
```

5. If you are using transactions (AutoCommit false in the iAnywhere.UltraLite namespace), commit the change.

```
// iAnywhere.Data.UltraLite namespace
cmd.Transaction.Commit();
```

```
// iAnywhere.UltraLite namespace
conn.Commit();
```

Data retrieval: SELECT

The SELECT statement allows you to retrieve information from the

database. This section describes how to execute a SELECT statement and how to handle the result set it returns.

❖ To execute a SELECT statement

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

```
// iAnywhere.Data.UltraLite namespace
ULCommand cmd;
```

```
// iAnywhere.UltraLite namespace
PreparedStatement prepStmt;
```

2. Assign a statement to the object.

```
// iAnywhere.Data.UltraLite namespace
cmd = conn.CreateCommand();
cmd.Command =
   "SELECT MyColumn FROM MyTable";
```

```
// iAnywhere.UltraLite namespace
prepStmt = conn.PrepareStatement(
"SELECT MyColumn FROM MyTable");
```

3. Execute the statement.

   In the following code, the result of the SELECT query contain a string, which is output to a command prompt.

```
// iAnywhere.Data.UltraLite namespace
ULDataReader customerNames = prepStmt.ExecuteReader();
while( customerNames.MoveNext() ) {
  for ( int i = 0;
    i < customerNames.GetFieldCount();
    i++ ) {
    System.Console.Write(
    customerNames.GetString( i ) + " " );
  }
  System.Console.WriteLine();
}
```

```
// iAnywhere.UltraLite namespace
ResultSet customerNames = prepStmt.ExecuteQuery();
customerNames.MoveBeforeFirst();
while( customerNames.MoveNext() ) {
  for ( int i = 1;
    i <= customerNames.Schema.GetColumnCount();
    i++ ) {
    System.Console.Write(
    customerNames.GetString( i ) + " " );
  }
  System.Console.WriteLine();
}
```

Navigating dynamic SQL result sets

You can navigate through a result set using methods associated with the ULDataReader (iAnywhere.Data.UltraLite namespace) or ResultSet (iAnywhere.UltraLite namespace) object.

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

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

Result set schema description

The ULDataReader.GetSchemaTable method and ULDataReader.Schema property (iAnywhere.Data.UltraLite namespace), or the ResultSet.Schema property (iAnywhere.UltraLite namespace) allows you to retrieve information about a result set, such as column names, total number of columns, column scales, column sizes and column SQL types.

Example

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

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

```
// iAnywhere.UltraLite namespace
for ( int i = 1;
      i <= MyResultSet.Schema.GetColumnCount();
      i++ ) {
   System.Console.WriteLine(
      MyResultSet.Schema.GetColumnName(i) + " " +
      MyResultSet.Schema.GetColumnSQLType(i)
   );
}
```

Accessing and manipulating data with the Table API

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

☞ For information about dynamic SQL, see "Accessing and manipulating data using dynamic SQL" on page 29.

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

- Scrolling through the rows of a table.
- Accessing the values of the current row.
- Using find and lookup methods to locate rows in a table.
- Inserting, deleting, and updating rows.

Navigating the rows of a table

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

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

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

Example

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

```
// iAnywhere.Data.UltraLite namespace
ULTable t = conn.ExecuteTable( "MyTable" );
int colID = t.GetOrdinal( "MyColumn" );
while ( t.MoveNext() ){
   System.Console.WriteLine( t.GetString( colID ) );
}
// iAnywhere.UltraLite namespace
Table t = conn.GetTable( "MyTable" );
short colID = t.Schema.GetColumnID( "MyColumn" );
t.Open();
t.MoveBeforeFirst();
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.

```
// iAnywhere.Data.UltraLite namespace
ULTable t = conn.ExecuteTable( "MyTable", "ix_col" );
t.MoveFirst();

// iAnywhere.UltraLite namespace
Table t = conn.GetTable( "MyTable" );
t.Open( "ix_col" );
t.MoveFirst();
```

☞ For more information, see "ULTable class" on page 289 and "ULTableSchema class" on page 324 and (iAnywhere.Data.UltraLite namespace) or "Table class" on page 546 and "TableSchema class" on page 585 (iAnywhere.UltraLite namespace).

Using UltraLite modes

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

- **Insert mode** The data in the buffer is added to the table as a new row when the insert method is called.
- **Update mode** The data in the buffer replaces the current row when the update method is called.
- **Find mode** Used to locate a row whose value exactly matches the data

in the buffer when one of the find methods is called.

- **Lookup mode** Used to locate a row whose value matches or is greater than the data in the buffer when one of the lookup methods is called.

Accessing the values of the current row

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

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

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

Retrieving column values

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

Example

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

```
// iAnywhere.Data.UltraLite namespace
int lname = t.GetOrdinal( "lname" );
string lastname = t.GetString( lname );
```

```
// iAnywhere.UltraLite namespace
short lname = t.Schema.GetColumnID( "lname" );
string lastname = t.GetString( lname );
```

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

```
// iAnywhere.Data.UltraLite namespace
int cust_id = t.GetOrdinal( "cust_id" );
int id = t.GetInt( cust_id );
```

```
// iAnywhere.UltraLite namespace
short cust_id = t.Schema.GetColumnID( "cust_id" );
int id = t.GetInt( cust_id );
```

Modifying column values

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

Example

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

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

By assigning values to these properties you do not alter the value of the data in the database. You can assign values to the properties even if you are

before the first row or after the last row of the table, but it is an error to try to access data when the current row is at one of these positions, for example by assigning the property to a variable.

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

Casting values

The method you choose must match the data type you wish 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.

Searching rows with find and lookup

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

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

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

❖ To search for a row

1. Enter find or lookup mode.

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

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

2. Set the search values.

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

```
// iAnywhere.Data.UltraLite namespace
int lname = t.GetOrdinal( "lname" );
t.SetString( lname, "Kaminski" );

// iAnywhere.UltraLite namespace
short lname = t.Schema.GetColumnID( "lname" );
t.SetString( lname, "Kaminski" );
```

3. Search for the row.

   Use the appropriate method to carry out the search. For example, the following instruction looks for the first row that exactly matches the specified value in the current index.

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

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

4. Search for the next instance of the row.

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

☞ For more information, see "ULTable class" on page 289 (iAnywhere.Data.UltraLite namespace) or "Table class" on page 546 (iAnywhere.UltraLite namespace).

Updating rows

The following procedure updates a row.

❖ To update a row

1. Move to the row you wish to update.

   You can move to a row by scrolling through the table or by searching the table using find and lookup methods.

2. Enter update mode.

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

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

3. Set the new values for the row to be updated. For example, the following instruction sets the id column in the buffer to 3.

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

4. Execute the Update.

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

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

By default, UltraLite.NET operates in autocommit mode, so that the update is immediately applied to the row in permanent storage. If you have disabled autocommit mode, the update is not applied until you execute a commit operation. For more information, see "Transaction processing in UltraLite" on page 43.

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

Inserting rows

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

Example

The following code inserts a new row.

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

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

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

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

Deleting rows

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

The following procedure deletes a row.

❖ To delete a row

1. Move to the row you wish to delete.
2. Execute the Table.Delete() method.

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

Transaction processing in UltraLite

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

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

To use multi-statement transactions, you must create a "ULTransaction class" on page 335 object by calling ULConnection.BeginTransaction(). For example, if your application transfers money between two accounts, either both the deduction from the source account and the addition to the destination account must be completed, or neither statement must be completed.

If the connection has a valid transaction, you must execute a ULTransaction.Commit() statement to complete the transaction and make changes to your database permanent, or you must execute a ULTransaction.Rollback() statement to cancel all the operations of the transaction. Once a transaction has been commited or rolled back, the connection will revert to autocommit mode until the next call to ULConnection.BeginTransaction().

☞ For more information, see "ULConnection class" on page 72 and "ULTransaction class" on page 335 (iAnywhere.Data.UltraLite namespace) or "Connection class" on page 345 (iAnywhere.UltraLite namespace).

Accessing schema information

The objects in the 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.

☞ For information about modifying the schema, see "Upgrading your database schema" on page 22.

You can access the following schema objects and information.

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

  To obtain a ULDatabaseSchema object, access ULConnection.Schema (iAnywhere.Data.UltraLite namespace) or Connection.Schema (iAnywhere.UltraLite namespace).

  ☞ For more information, see "ULConnection class" on page 72 (iAnywhere.Data.UltraLite namespace) or "Connection class" on page 345 (iAnywhere.UltraLite namespace).

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

  To obtain a ULTableSchema object, access ULTable.Schema (iAnywhere.Data.UltraLite namespace) or Table.Schema (iAnywhere.UltraLite namespace).

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

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

- **PublicationSchema** A list of the tables and columns contained in a publication. Publications are also comprised of schema only, and so there is no Publication object.

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

☞ For more information, see "ULTableSchema class" on page 324 (iAnywhere.Data.UltraLite namespace) or "TableSchema class" on page 585 (iAnywhere.UltraLite namespace).

Error handling

You can use the standard .NET error-handling features to handle errors. Most iAnywhere.Data.UltraLite namespace methods throw ULException errors. You can use ULException.NativeError to retrieve the ULSQLCode value assigned to this error. Most iAnywhere.UltraLite namespace methods throw SQLException errors. You can use SQLException.ErrorCode to retrieve the SQLCode value assigned to this error. Both ULException and SQLException have a Message property which you can use to obtain a descriptive text of the error. ULSQLCode and SQLCode errors are negative numbers indicating the error type.

☞ For a list of error codes, see *ASA Error Messages*.

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

☞ For more information, see the following:

- "ULSyncProgressListener interface" on page 281 (iAnywhere.Data.UltraLite namespace) or "SyncProgressListener interface" on page 539 (iAnywhere.UltraLite namespace).
- "ULSyncResult class" on page 284 (iAnywhere.Data.UltraLite namespace) or "SyncResult class" on page 542 (iAnywhere.UltraLite namespace).

User authentication

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

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

☞ For more information, see "User authentication in UltraLite" [*UltraLite Database User's Guide*, page 40].

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

1. Connect to the database as a user with DBA authority.
2. Grant the user access to the database with the desired password using the ULConnection.GrantConnectTo method.

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

   ☞ For more information, see "ULConnection class" on page 72 (iAnywhere.Data.UltraLite namespace) or "Connection class" on page 345 (iAnywhere.UltraLite namespace).

❖ To delete an existing user

1. Connect to the database as a user with DBA authority.
2. Revoke the user's connection authority using the Connection.RevokeConnectFrom method.

Synchronizing UltraLite applications

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

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

☞ For a more detailed explanation of synchronization, see "UltraLite Clients" [*MobiLink Clients*, page 277].

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 9 installation.

UltraLite.NET supports TCP/IP, HTTP, and HTTPS synchronization. Synchronization is initiated by the UltraLite application. In all cases, you use properties of the SyncParms object to control synchronization.

Note

To synchronize using HTTPS you must obtain the separately-licensable security option. To order this option, see the card in your SQL Anywhere Studio package or see *http://www.sybase.com/detail?id=1015780*.

☞ For more information, see "Welcome to SQL Anywhere Studio" [*Introducing SQL Anywhere Studio*, page 4].

Adding ActiveSync synchronization to your application

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

For general information about setting up ActiveSync synchronization, see "Deploying applications that use ActiveSync" [*MobiLink Clients*, page 312]. For general information about adding synchronization to an application, see "UltraLite Clients" [*MobiLink Clients*, page 277].

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

When ActiveSync initiates synchronization, the MobiLink ActiveSync provider starts the UltraLite application, if it is not already running, and

sends a message to it. Your application must implement a ULActiveSyncListener to receive and process messages from the MobiLink provider. Your application must specify the listener object using the SetActiveSyncListener method, where MyAppClassName is a unique Windows class name for the application.

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

For more information, including sample code, see the "ULActiveSyncListener interface" on page 52 (iAnywhere.Data.UltraLite namespace) or "ActiveSyncListener interface" on page 341 (iAnywhere.UltraLite namespace).

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

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

When registering your application, set the following parameter.

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

CHAPTER 4

iAnywhere.Data.UltraLite namespace

About this chapter

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

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

UltraLite.NET extensions that are not available in Adaptive Server Anywhere's Data Provider for ADO.NET are denoted in this API reference with "**UL Ext.:**".

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

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

The **iAnywhere.Data.UltraLite** assembly uses a satellite resource assembly called **iAnywhere.Data.UltraLite.resources**. The main assembly searches for this resource assembly by culture, using the following order: System.Globalization.CultureInfo.CurrentUICulture, then System.Globalization.CultureInfo.CurrentCulture, and finally culture "EN".

Contents

ULActiveSyncListener interface

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

Prototypes

' Visual Basic
Public Interface **ULActiveSyncListener**

// C#
public interface **ULActiveSyncListener**

ULActiveSyncListener members

Public instance methods

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

ActiveSyncInvoked method

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

Prototypes

' Visual Basic
Public Sub **ActiveSyncInvoked(** _
  ByVal *launchedByProvider* As Boolean _
)

// C#
public void **ActiveSyncInvoked(**
  bool *launchedByProvider*
);

Parameters

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

Remarks

This method is invoked by a separate thread. To avoid multi-threading issues, it should post an event to the UI. If you are using multi-threading, it is recommended that you use a separate connection and use the lock keyword to access any objects shared with the rest of the application. The synchronization resulting from a call to ActiveSyncInvoked must specify the ULStreamType enumeration value for its connection's Stream property property.

Example

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

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

Public Class MainWindow
  Inherits System.Windows.Forms.Form
  Implements ULActiveSyncListener
  Private conn As ULConnection
  Public Sub New(ByVal args() As String)

    MyBase.New()

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

    'Add any initialization after the InitializeComponent() call
    ULConnection.DatabaseManager.SetActiveSyncListener( _
        "myCompany.myapp", Me _
      )
    'Create Connection
    ...
  End Sub
  Protected Overrides Sub OnClosing( _
        ByVal e As System.ComponentModel.CancelEventArgs _
      )
    ULConnection.DatabaseManager.SetActiveSyncListener( _
        Nothing, Nothing _
      )
    MyBase.OnClosing(e)
  End Sub
  Public Sub ActiveSyncInvoked( _
        ByVal launchedByProvider As Boolean _
      ) Implements ULActiveSyncListener.ActiveSyncInvoked
    Me.Invoke(New EventHandler(AddressOf Me.ActiveSyncAction))
  End Sub
  Public Sub ActiveSyncAction( _
        ByVal sender As Object, ByVal e As EventArgs _
      )
    ' Do active sync
    conn.SyncParms.Stream = ULStreamType.ACTIVE_SYNC
    conn.Synchronize()
  End Sub
End Class
// C#
using iAnywhere.Data.UltraLite;
public class Form1 : System.Windows.Forms.Form,
         ULActiveSyncListener
{
  private System.Windows.Forms.MainMenu mainMenu1;
  private ULConnection conn;
```

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

    //
    // TODO: Add any constructor code after
    // InitializeComponent call
    //
    ULConnection.DatabaseManager.SetActiveSyncListener(
        "myCompany.myapp", this
      );
    // Create connection
    ...
  }
  protected override void Dispose( bool disposing )
  {
    base.Dispose( disposing );
  }
  protected override void OnClosing(
      System.ComponentModel.CancelEventArgs e
    )
  {
    ULConnection.DatabaseManager.SetActiveSyncListener(
        null, null
      );
    base.OnClosing(e);
  }
  public void ActiveSyncInvoked(bool launchedByProvider)
  {
    this.Invoke( new EventHandler( ActiveSyncHandler ) );
  }
  internal void ActiveSyncHandler(object sender, EventArgs e)
  {
    conn.SyncParms.Stream = ULStreamType.ACTIVE_SYNC;
    conn.Synchronize();
  }
}
```

ULAuthStatusCode enumeration

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

Prototypes

' Visual Basic
Public Enum **ULAuthStatusCode**

// C#
public enum **ULAuthStatusCode**

Members

Member	Description
EXPIRED	User ID or password has expired - authorization failed (EXPIRED = 3000).
IN_USE	User ID is already in use - authorization failed (IN_USE = 5000).
INVALID	Bad user ID or password - authorization failed (INVALID = 4000).
UNKNOWN	Authorization status is unknown, possibly because the connection has not yet performed a synchronization (UNKNOWN = 0).
VALID	User ID and password were valid at time of synchronization (VALID = 1000).
VALID_BUT_EXPIRES_SOON	User ID and password were valid at time of synchronization, but will expire soon (VALID_BUT_EXPIRES_SOON = 2000).

See also

- "AuthStatus property" on page 285

ULCommand class

Represents a pre-compiled SQL statement or query, with or without IN parameters. This object can be used to efficiently execute a statement or query multiple times.

Prototypes

```
' Visual Basic
NotInheritable Public Class ULCommand
  Inherits Component, IDbCommand
```

```
// C#
public sealed class ULCommand :
  Component,
  IDbCommand
```

Remarks

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

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

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

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

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

For resource management reasons, it is recommended that you explicitly

close commands when you are done with them.

Implements: IDbCommand, IDisposable

ULCommand members

Public instance constructors

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

Public instance properties

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

Public instance methods

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

Protected instance methods

Member	Description
Dispose method	Releases the unmanaged resources used by the ULCommand and optionally releases the managed resources.

ULCommand constructor

Initializes a ULCommand object.

Prototypes

' Visual Basic
Overloads Public Sub **New()**

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

Remarks

The ULCommand object needs to have the CommandText property, Connection property, and Transaction property properties set before a

statement can be executed.

See also

- "ULCommand class" on page 56
- "ULCommand members" on page 57
- "CreateCommand method" on page 86
- "ULCommand constructor" on page 59
- "ULCommand constructor" on page 59
- "ULCommand constructor" on page 60

ULCommand constructor

Initializes a ULCommand object with the specified command text.

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal cmdText As String _
)

// C#
public ULCommand(
  string  cmdText
);
```

Parameters

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

Remarks

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

See also

- "ULCommand class" on page 56
- "ULCommand members" on page 57
- "CreateCommand method" on page 86
- "ULCommand constructor" on page 58
- "ULCommand constructor" on page 59
- "ULCommand constructor" on page 60

ULCommand constructor

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

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal cmdText As String, _
  ByVal connection As ULConnection _
)
```

```
// C#
public ULCommand(
  string cmdText,
  ULConnection connection
);
```

Parameters

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

Remarks

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

See also

- "ULCommand class" on page 56
- "ULCommand members" on page 57
- "CreateCommand method" on page 86
- "ULCommand constructor" on page 58
- "ULCommand constructor" on page 59
- "ULCommand constructor" on page 60

ULCommand constructor

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

Prototypes

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

```
// C#
public ULCommand(
  string cmdText,
  ULConnection connection,
  ULTransaction transaction
);
```

Parameters

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

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

See also

- "ULCommand class" on page 56
- "ULCommand members" on page 57
- "CreateCommand method" on page 86
- "ULCommand constructor" on page 58
- "ULCommand constructor" on page 59
- "ULCommand constructor" on page 59

CommandText property

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

Prototypes

```
' Visual Basic
NotOverridable Public Property CommandText As String _
  Implements IDbCommand.CommandText

// C#
public string  CommandText {get;set;}
```

Property value

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

Implements

IDbCommand.CommandText

Example

The following example demonstrates the use of the parameterized placeholder:

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

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

See also

- "ULCommand class" on page 56
- "ULCommand members" on page 57
- "ExecuteNonQuery method" on page 66
- "ExecuteReader method" on page 66
- "ExecuteScalar method" on page 68
- "ExecuteTable method" on page 69

CommandTimeout property

This feature is not supported by UltraLite.NET.

Prototypes

```
' Visual Basic
NotOverridable Public Property CommandTimeout As Integer _
  Implements IDbCommand.CommandTimeout
```

```
// C#
public int CommandTimeout {get;set;}
```

Property value

The value is always zero.

Exceptions

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

Implements

IDbCommand.CommandTimeout

CommandType property

Specifies the type of command to be executed.

Prototypes

```
' Visual Basic
NotOverridable Public Property CommandType As CommandType _
  Implements IDbCommand.CommandType
```

```
// C#
public CommandType CommandType {get;set;}
```

Property value

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

Remarks

Supported command types are as follows:

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

Exceptions

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

Implements

IDbCommand.CommandType

Connection property

The connection object on which to execute the ULCommand object.

Prototypes

```
' Visual Basic
Public Property Connection As ULConnection
```

```
// C#
public ULConnection Connection {get;set;}
```

Property value — The ULConnection class object on which to execute the command.

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

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

This is the strongly typed version of IDbCommand.Connection.

IndexName property

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

Prototypes

```
' Visual Basic
Public Property IndexName As String

// C#
public string  IndexName {get;set;}
```

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

See also

- "ULCommand class" on page 56
- "ULCommand members" on page 57
- "ExecuteTable method" on page 69
- "ExecuteReader method" on page 66

Parameters property

Specifies the parameters for the current statement.

Prototypes

```
' Visual Basic
Public Readonly Property Parameters As ULParameterCollection

// C#
public ULParameterCollection Parameters {get;}
```

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

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

This is the strongly typed version of IDbCommand.Parameters.

Plan property

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

Prototypes

```
' Visual Basic
Public Readonly Property Plan As String

// C#
public string  Plan {get;}
```

Property value

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

Exceptions

- ULException class - A SQL error occurred.

Transaction property

Specifies the ULTransaction class in which the ULCommand executes.

Prototypes

```
' Visual Basic
Public Property Transaction As ULTransaction

// C#
public ULTransaction Transaction {get;set;}
```

Property value

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

Remarks

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

This is the strongly typed version of IDbCommand.Transaction.

See also

- “ULCommand class” on page 56
- “ULCommand members” on page 57
- “BeginTransaction method” on page 82

UpdatedRowSource property

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

Prototypes

```
' Visual Basic
NotOverridable Public Property UpdatedRowSource As UpdateRowSource _
  Implements IDbCommand.UpdatedRowSource

// C#
public UpdateRowSource UpdatedRowSource {get;set;}
```

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

Implements — IDbCommand.UpdatedRowSource

Cancel method

This method is not supported in UltraLite.NET.

Prototypes

```
' Visual Basic
NotOverridable Public Sub Cancel() _
  Implements IDbCommand.Cancel
```

```
// C#
public void Cancel();
```

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

Implements — IDbCommand.Cancel

CreateParameter method

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

Prototypes

```
' Visual Basic
Public Function CreateParameter() As ULParameter
```

```
// C#
public ULParameter CreateParameter();
```

Return value — A new parameter, as a ULParameter class object.

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

This is the strongly typed version of IDbCommand.CreateParameter.

Dispose method

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

Prototypes

```
' Visual Basic
Overloads Overrides Protected Sub Dispose( _
  ByVal disposing As Boolean _
)
```

```
// C#
protected override void Dispose(
  bool disposing
);
```

Parameters

- **disposing** When true, disposes of both the managed and unmanaged resources. When false, disposes of only the unmanaged resources.

ExecuteNonQuery method

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

Prototypes

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

```
// C#
public int ExecuteNonQuery();
```

Return value

The number of rows affected.

Remarks

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

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

The CommandType property property cannot be CommandType.TableDirect.

Exceptions

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

Implements

IDbCommand.ExecuteNonQuery

ExecuteReader method

Executes a SQL SELECT statement and returns the result set.

Prototypes

```
' Visual Basic
Overloads Public Function ExecuteReader() As ULDataReader
```

```
// C#
public ULDataReader ExecuteReader();
```

Return value

The result set as a ULDataReader class object.

Remarks

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

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

This is the strongly typed version of IDbCommand.ExecuteReader.

Exceptions

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

See also

- "ULCommand class" on page 56
- "ULCommand members" on page 57
- "ExecuteReader method" on page 67

ExecuteReader method

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

Prototypes

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

```
// C#
public ULDataReader ExecuteReader(
  CommandBehavior cmdBehavior
);
```

Parameters

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

Return value

The result set as a ULDataReader class object.

Remarks

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

object is a read-only result set. For editable result sets, use ExecuteTable method or a ULDataAdapter class.

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

This is the strongly typed version of IDbCommand.ExecuteReader.

Exceptions

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

See also

- "ULCommand class" on page 56
- "ULCommand members" on page 57
- "ExecuteReader method" on page 66

ExecuteScalar method

Executes a SQL SELECT statement and returns a single value.

Prototypes

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

```
// C#
public object ExecuteScalar();
```

Return value

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

Remarks

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

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

If the CommandType property is CommandType.TableDirect, ExecuteScalar performs an ExecuteTable method and returns the first column of the first row.

Exceptions

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

Implements IDbCommand.ExecuteScalar

ExecuteTable method

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

Prototypes

```
' Visual Basic
Overloads Public Function ExecuteTable() As ULTable

// C#
public ULTable ExecuteTable();
```

Return value The table as a ULTable class object.

Remarks The CommandType property property must be set to CommandType.TableDirect.

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

Exceptions

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

See also

- “ULCommand class” on page 56
- “ULCommand members” on page 57
- “ExecuteTable method” on page 69

ExecuteTable method

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

Prototypes

```
' Visual Basic
Overloads Public Function ExecuteTable( _
  ByVal cmdBehavior As CommandBehavior _
) As ULTable
```

```
// C#
public ULTable ExecuteTable(
  CommandBehavior cmdBehavior
);
```

Parameters

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

Return value

The table as a ULTable class object.

Remarks

The CommandType property property must be set to CommandType.TableDirect.

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

Exceptions

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

See also

- "ULCommand class" on page 56
- "ULCommand members" on page 57
- "ExecuteTable method" on page 69

Prepare method

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

Prototypes

```
' Visual Basic
NotOverridable Public Sub Prepare() _
  Implements IDbCommand.Prepare
```

```
// C#
public void Prepare();
```

Remarks

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

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

Exceptions

- ULException class - A SQL error occurred.

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

Implements IDbCommand.Prepare

ULConnection class

Represents a connection to an UltraLite.NET database.

Prototypes

```
' Visual Basic
NotInheritable Public Class ULConnection
  Inherits Component, IDbConnection

// C#
public sealed class ULConnection :
  Component,
  IDbConnection
```

Remarks

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

A connection to an existing database is opened using the Open method method. You can create a new database and connect to it using the OpenWithCreate method method.

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

The schema of the database can be accessed using an open connection's Schema property property.

Implements: IDbConnection, IDisposable

ULConnection members

Public static fields
(Shared)

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

Public static properties
(Shared)

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

Public instance constructors

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

Public instance properties

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

Public instance methods

Member	Description
BeginTransaction method	Returns a transaction object. Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a Commit method or Rollback method.
BeginTransaction method	Returns a transaction object with the specified isolation level. Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a Commit method or Rollback method.
ChangeDatabase method	Changes the current database for an open ULConnection.
ChangeEncryptionKey method	**UL Ext.:** Changes the database's encryption key to the specified new key.
Close method	Closes the database connection.
CountUploadRows method	**UL Ext.:** Returns the number of rows that need to be uploaded when the next synchronization takes place.
CountUploadRows method	**UL Ext.:** Returns the number of rows that need to be uploaded when the next synchronization takes place.
CreateCommand method	Creates and initializes a ULCommand class object associated with this connection and its current transaction. You can use the properties of the ULCommand to control its behavior.
ExecuteTable method	**UL Ext.:** Retrieves in a ULTable class a database table for direct manipulation. The table is opened (sorted) using the table's primary key.
ExecuteTable method	**UL Ext.:** Retrieves in a ULTable class a database table for direct manipulation. The table is opened (sorted) using the specified index.
ExecuteTable method	**UL Ext.:** Retrieves, with the specified command behavior, a database table for direct manipulation. The table is opened (sorted) using the specified index.
GetLastDownloadTime method	**UL Ext.:** Returns the time of the most recent download of the specified publication.
GetNewUUID method	**UL Ext.:** Generates a new UUID (Guid).
GrantConnectTo method	**UL Ext.:** Grants access to an UltraLite database for a user ID with a specified password.
Open method	Opens a connection to a database using the previously-specified connection string.

Member	Description
OpenWithCreate method	**UL Ext.:** Opens a connection to a database using the previously-specified connection string. If the database does not exist, a new database is created.
ResetLastDownloadTime method	**UL Ext.:** Resets the time of the most recent download.
RevokeConnectFrom method	**UL Ext.:** Revokes access to an UltraLite database from the specified user ID.
RollbackPartialDownload method	**UL Ext.:** Rolls back outstanding changes to the database from a partial download.
StartSynchronizationDelete method	**UL Ext.:** Marks all subsequent deletes made by this connection for synchronization.
StopSynchronizationDelete method	**UL Ext.:** Prevents delete operations from being synchronized.
Synchronize method	**UL Ext.:** Synchronize the database using the current SyncParms property.
Synchronize method	**UL Ext.:** Synchronize the database using the current SyncParms property with progress events posted to the specified listener.

Public instance events

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

Protected instance methods

Member	Description
Dispose method	Releases the unmanaged resources used by the ULCommand class and optionally releases the managed resources.
Finalize method	Releases unmanaged resources and performs other cleanup operations before the ULConnection is reclaimed by garbage collection.

ULConnection constructor

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

Prototypes

```
' Visual Basic
Overloads Public Sub New()
```

```
// C#
public ULConnection();
```

Remarks

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

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

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "Open method" on page 93
- "OpenWithCreate method" on page 93
- "ULConnection constructor" on page 76

ULConnection constructor

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

Prototypes

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

```
// C#
public ULConnection(
  string  connectionString
);
```

Parameters

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

  For a list of parameters, see the ConnectionString property property.

Remarks

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

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

Exceptions

- ArgumentException - The supplied connection string is invalid.

Example

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

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

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

See also

- “ULConnection class” on page 72
- “ULConnection members” on page 72
- “Open method” on page 93
- “OpenWithCreate method” on page 93
- “ULConnection constructor” on page 76

INVALID_DATABASE_ID field

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

Prototypes

' Visual Basic
Public Shared **INVALID_DATABASE_ID** As Long

// C#
public const long **INVALID_DATABASE_ID**;

ConnectionString property

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

Prototypes

' Visual Basic
NotOverridable Public Property **ConnectionString** As String _
  Implements IDbConnection.ConnectionString

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

Property value

The parameters used to open this connection in the form of a semicolon-separated list of keyword-value pairs. The default is an empty

string (an invalid connection string).

Remarks

UL Ext.: The parameters used by UltraLite.NET are specific to UltraLite databases and therefore the connection string is not compatible with Adaptive Server Anywhere connection strings. For a list of parameters, see "Connection Parameters" [*UltraLite Database User's Guide*, page 63].

Exceptions

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

Implements

IDbConnection.ConnectionString

Example

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

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

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

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "Open method" on page 93
- "OpenWithCreate method" on page 93

ConnectionTimeout property

This feature is not supported by UltraLite.NET.

Prototypes

' Visual Basic

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

// C#

public int **ConnectionTimeout** {get;}

Property value

The value is always zero.

Exceptions

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

Implements

IDbConnection.ConnectionTimeout

Database property

Returns the name of the database that the connection will be opened to.

Prototypes

' Visual Basic
NotOverridable Public Readonly Property **Database** As String _
Implements IDbConnection.Database

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

Property value

A string containing the name of the database.

Remarks

On Windows CE devices, ULConnection looks in the connection string in the following order: dbn, ce_file.

On desktop machines, ULConnection looks in the connection string in the following order: dbn, nt_file.

Implements

IDbConnection.Database

DatabaseID property

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

Prototypes

' Visual Basic
Public Property **DatabaseID** As Long

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

Property value

The Database ID value of the current database.

Remarks

The database ID value must be in the range [0,UInt32.MaxValue]. A value of INVALID_DATABASE_ID field is used to indicate that the database ID has not been set for the current database.

Exceptions

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

DatabaseManager property

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

Prototypes

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

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

Property value — A reference to the singleton ULDatabaseManager class object.

GlobalAutoIncrementUsage property

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

Prototypes

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

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

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

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

Exceptions

- ULException class - A SQL error occurred.

LastIdentity property

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

Prototypes

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

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

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

Remarks — The most recent identity value used. This property is equivalent to the Adaptive Server Anywhere SQL statement:

```
SELECT @@identity
```

LastIdentity is particularly useful in the context of global autoincrement columns.

Since this property only allows you to determine the most recently assigned default value, you should retrieve this value soon after executing the insert statement to avoid spurious results.

Occasionally, a single insert statement may include more than one column of type global autoincrement. In this case, LastIdentity is one of the generated default values, but there is no reliable means to determine which column the value is from. For this reason, you should design your database and write your insert statements so as to avoid this situation.

Exceptions

- ULException class - A SQL error occurred.

Schema property

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

Prototypes

```
' Visual Basic
Public Readonly Property Schema As ULDatabaseSchema

// C#
public ULDatabaseSchema Schema {get;}
```

Property value

A reference to the ULDatabaseSchema class object representing the schema of the database that this connection is opened on.

Remarks

This property is only valid while its connection is open.

State property

Returns the current state of the connection.

Prototypes

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

// C#
public ConnectionState State {get;}
```

Property value

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

Implements

IDbConnection.State

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "StateChange event" on page 98

SyncParms property

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

Prototypes

```
' Visual Basic
Public Readonly Property SyncParms As ULSyncParms

// C#
public ULSyncParms SyncParms {get;}
```

Property value

A reference to the parameters used for synchronization by this connection. Modifications to the parameters affect the next synchronization made over

this connection.

See also

- “ULConnection class” on page 72
- “ULConnection members” on page 72
- “Synchronize method” on page 96
- “SyncResult property” on page 82

SyncResult property

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

Prototypes

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

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

Property value

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

See also

- “ULConnection class” on page 72
- “ULConnection members” on page 72
- “Synchronize method” on page 96
- “SyncParms property” on page 81

BeginTransaction method

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

Prototypes

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

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

Return value

A ULTransaction class object representing the new transaction.

Remarks

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

By default, the connection does not use transactions and all commands are automatically committed as they are executed. Once the current transaction is committed or rolled back, the connection reverts to auto commit mode until the next call to BeginTransaction.

This is the strongly typed version of IDbConnection.BeginTransaction.

Exceptions

- ULException class - The connection is closed.

- InvalidOperationException - ULConnection does not support parallel transactions.

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "BeginTransaction method" on page 83

BeginTransaction method

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

Prototypes

```
' Visual Basic
Overloads Public Function BeginTransaction( _
  ByVal isolationLevel As IsolationLevel _
) As ULTransaction
```

```
// C#
public ULTransaction BeginTransaction(
  IsolationLevel isolationLevel
);
```

Parameters

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

Return value

A ULTransaction class object representing the new transaction.

Remarks

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

By default, the connection does not use transactions and all commands are automatically committed as they are executed. Once the current transaction is committed or rolled back, the connection reverts to auto commit mode until the next call to BeginTransaction.

This is the strongly typed version of IDbConnection.BeginTransaction.

Exceptions

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

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "BeginTransaction method" on page 82

ChangeDatabase method

Changes the current database for an open ULConnection.

Prototypes

```
' Visual Basic
NotOverridable Public Sub ChangeDatabase( _
  ByVal connectionString As String _
) _
  Implements IDbConnection.ChangeDatabase
```

```
// C#
public void ChangeDatabase(
  string  connectionString
);
```

Parameters

- **connectionString** A complete connection string to open the connection to a new database.

Remarks

The connection to the current database is closed even if there are parameter errors.

UL Ext.: *connectionString* is a full connection string, not a dbn or dbf.

Implements

IDbConnection.ChangeDatabase

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "ConnectionString property" on page 77

ChangeEncryptionKey method

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

Prototypes

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

```
// C#
public void ChangeEncryptionKey(
  string  newKey
);
```

Parameters

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

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 72

- "ULConnection members" on page 72
- "EncryptionKey property" on page 107

Close method

Closes the database connection.

Prototypes

```
' Visual Basic
NotOverridable Public Sub Close() _
  Implements IDbConnection.Close
```

```
// C#
public void Close();
```

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

Implements

IDbConnection.Close

CountUploadRows method

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

Prototypes

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

```
// C#
public uint CountUploadRows(
  int mask,
  uint threshold
);
```

Parameters

- **mask** The set of publications to check for rows. See the ULPublicationSchema class class for more information.
- **threshold** The maximum number of rows to count, limiting the amount of time taken by CountUploadRows. A value of 0 corresponds to the maximum limit. A value of 1 determines if any rows need to be synchronized.

Return value

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

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "CountUploadRows method" on page 86

CountUploadRows method

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

Prototypes

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

```
// C#
public long CountUploadRows(
  int mask,
  long threshold
);
```

Parameters

- **mask** The set of publications to check for rows. See the ULPublicationSchema class class for more information.
- **threshold** The maximum number of rows to count, limiting the amount of time taken by CountUploadRows. A value of 0 corresponds to the maximum limit. A value of 1 determines if any rows need to be synchronized. The threshold value must be in the range [0,0x0fffffff].

Return value

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

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "CountUploadRows method" on page 85

CreateCommand method

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

Prototypes

```
' Visual Basic
Public Function CreateCommand() As ULCommand
```

```
// C#
public ULCommand CreateCommand();
```

Return value

A new ULCommand class object.

Remarks

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

This is the strongly typed version of IDbConnection.CreateCommand.

Dispose method

Releases the unmanaged resources used by the ULCommand class and optionally releases the managed resources.

Prototypes

```
' Visual Basic
Overloads Overrides Protected Sub Dispose( _
  ByVal disposing As Boolean _
)
```

```
// C#
protected override void Dispose(
  bool disposing
);
```

Parameters

- **disposing** When true, dispose of both managed and unmanaged resources. When false, dispose of only the unmanaged resources.

ExecuteTable method

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

Prototypes

```
' Visual Basic
Overloads Public Function ExecuteTable( _
  ByVal tableName As String _
) As ULTable
```

```
// C#
public ULTable ExecuteTable(
  string tableName
);
```

Parameters

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

Return value

The table as a ULTable class object.

Remarks

This method is a shortcut for the ExecuteTable method method that does not require a ULCommand class instance. It is provided to help users porting from earlier versions of UltraLite.NET (it replaces

iAnywhere.UltraLite.Connection.GetTable() and iAnywhere.UltraLite.Table.Open()).

Exceptions

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

Example

The following code opens the table named MyTable using the table's primary key. It assumes an open ULConnection instance called conn.

```
' Visual Basic
Dim t As ULTable = conn.ExecuteTable("MyTable")
' The line above is equivalent to
' Dim cmd As ULCommand = conn.CreateCommand()
' cmd.CommandText = "MyTable"
' cmd.CommandType = CommandType.TableDirect
' Dim t As ULTable = cmd.ExecuteTable()
' cmd.Dispose()
```

```
// C#
ULTable t = conn.ExecuteTable("MyTable");
// The line above is equivalent to
// ULTable t;
// using(ULCommand cmd = conn.CreateCommand())
// {
//     cmd.CommandText = "MyTable";
//     cmd.CommandType = CommandType.TableDirect;
//     t = cmd.ExecuteTable();
// }
```

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "ExecuteTable method" on page 88
- "ExecuteTable method" on page 89

ExecuteTable method

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

Prototypes

```
' Visual Basic
Overloads Public Function ExecuteTable( _
  ByVal tableName As String, _
  ByVal indexName As String _
) As ULTable
```

```
// C#
public ULTable ExecuteTable(
  string  tableName,
  string  indexName
);
```

Parameters

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

- **indexName** The name of the index to open (sort) the table with.

Return value

The table as a ULTable class object.

Remarks

This method is a shortcut for the ExecuteTable method method that does not require a ULCommand class instance. It is provided to help users porting from earlier versions of UltraLite.NET (it replaces iAnywhere.UltraLite.Connection.GetTable() and iAnywhere.UltraLite.Table.Open()).

Exceptions

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

Example

The following code opens the table named MyTable using the index named MyIndex. It assumes an open ULConnection instance called conn.

```
' Visual Basic
Dim t As ULTable = conn.ExecuteTable("MyTable", "MyIndex")
' The line above is equivalent to
' Dim cmd As ULCommand = conn.CreateCommand()
' cmd.CommandText = "MyTable"
' cmd.IndexName = "MyIndex"
' cmd.CommandType = CommandType.TableDirect
' Dim t As ULTable = cmd.ExecuteTable()
' cmd.Dispose()
```

```
// C#
ULTable t = conn.ExecuteTable("MyTable", "MyIndex");
// The line above is equivalent to
// ULTable t;
// using(ULCommand cmd = conn.CreateCommand())
// {
//     cmd.CommandText = "MyTable";
//     cmd.IndexName = "MyIndex";
//     cmd.CommandType = CommandType.TableDirect;
//     t = cmd.ExecuteTable();
// }
```

See also

- “ULConnection class” on page 72
- “ULConnection members” on page 72
- “ExecuteTable method” on page 87
- “ExecuteTable method” on page 89

ExecuteTable method

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

Prototypes

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

// C#
public ULTable ExecuteTable(
  string tableName,
  string indexName,
  CommandBehavior cmdBehavior
);
```

Parameters

- **tableName** The name of the table to open.
- **indexName** The name of the index to open (sort) the table with.
- **cmdBehavior** A bitwise combination of CommandBehavior flags describing the results of the query and its effect on the connection. UltraLite.NET respects only the CommandBehavior.Default, CommandBehavior.CloseConnection, and CommandBehavior.SchemaOnly flags.

Return value

The table as a ULTable class object.

Remarks

This method is a shortcut for the ExecuteTable method method that does not require a ULCommand class instance. It is provided to help users porting from earlier versions of UltraLite.NET (it replaces iAnywhere.UltraLite.Connection.GetTable() and iAnywhere.UltraLite.Table.Open()).

Exceptions

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

Example

The following code opens the table named MyTable using the index named MyIndex. It assumes an open ULConnection instance called conn.

```
' Visual Basic
Dim t As ULTable = conn.ExecuteTable( _
    "MyTable", "MyIndex", CommandBehavior.Default _
  )
' The line above is equivalent to
' Dim cmd As ULCommand = conn.CreateCommand()
' cmd.CommandText = "MyTable"
' cmd.IndexName = "MyIndex"
' cmd.CommandType = CommandType.TableDirect
' Dim t As ULTable = cmd.ExecuteTable(CommandBehavior.Default)
' cmd.Dispose()
```

```
// C#
ULTable t = conn.ExecuteTable(
    "MyTable", "MyIndex", CommandBehavior.Default
  );
// The line above is equivalent to
// ULTable t;
// using(ULCommand cmd = conn.CreateCommand())
// {
//     cmd.CommandText = "MyTable";
//     cmd.IndexName = "MyIndex";
//     cmd.CommandType = CommandType.TableDirect;
//     t = cmd.ExecuteTable(CommandBehavior.Default);
// }
```

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "ExecuteTable method" on page 87
- "ExecuteTable method" on page 88

Finalize method

Releases unmanaged resources and performs other cleanup operations before the ULConnection is reclaimed by garbage collection.

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

GetLastDownloadTime method

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

Prototypes

' Visual Basic
Public Function **GetLastDownloadTime(** _
  ByVal *mask* As Integer _
) As Date

// C#
public DateTime **GetLastDownloadTime(**
  int *mask*
);

Parameters

- **mask** The mask of the publication to check. See the ULPublicationSchema class class for more information.

Return value

The timestamp of the last download.

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "ResetLastDownloadTime method" on page 94

GetNewUUID method

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

Prototypes

```
' Visual Basic
Public Function GetNewUUID() As Guid
```

```
// C#
public Guid GetNewUUID();
```

Return value

A new UUID as a Guid.

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

GrantConnectTo method

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

Prototypes

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

```
// C#
public void GrantConnectTo(
  string uid,
  string pwd
);
```

Parameters

- **uid** The user ID to receive access to the database. The maximum length of the user ID is 16 characters.
- **pwd** The password to be associated with the user ID. The maximum length is 16 characters.

Remarks

If an existing user ID is specified, this function updates the password for the user. UltraLite supports a maximum of 4 users. This method is enabled only if user authentication was enabled when the connection was opened.

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "UserID property" on page 109
- "Password property" on page 108
- "ConnectionString property" on page 77

Open method

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

Prototypes

```
' Visual Basic
NotOverridable Public Sub Open() _
  Implements IDbConnection.Open

// C#
public void Open();
```

Remarks

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

Exceptions

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

Implements

IDbConnection.Open

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "ConnectionString property" on page 77
- "OpenWithCreate method" on page 93
- "State property" on page 81

OpenWithCreate method

UL Ext.: Opens a connection to a database using the previously-specified connection string. If the database does not exist, a new database is created.

Prototypes

```
' Visual Basic
Public Sub OpenWithCreate()

// C#
public void OpenWithCreate();
```

Remarks

To create a new database, the connection string must contain the location of the new database's schema file.

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

Exceptions

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

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "ConnectionString property" on page 77
- "SchemaOnCE property" on page 108
- "SchemaOnDesktop property" on page 108
- "Open method" on page 93
- "State property" on page 81
- "DropDatabase method" on page 138

ResetLastDownloadTime method

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

Prototypes

```
' Visual Basic
Public Sub ResetLastDownloadTime( _
  ByVal mask As Integer _
)

// C#
public void ResetLastDownloadTime(
  int mask
);
```

Parameters

- **mask** The set of publications to reset. See ULPublicationSchema class class for more information.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "GetLastDownloadTime method" on page 91

RevokeConnectFrom method

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

Prototypes

```
' Visual Basic
Public Sub RevokeConnectFrom( _
  ByVal uid As String _
)

// C#
public void RevokeConnectFrom(
  string uid
);
```

Parameters

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

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "GrantConnectTo method" on page 92

RollbackPartialDownload method

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

Prototypes

```
' Visual Basic
Public Sub RollbackPartialDownload()

// C#
public void RollbackPartialDownload();
```

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "KeepPartialDownload property" on page 266
- "ResumePartialDownload property" on page 269

StartSynchronizationDelete method

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

Prototypes

```
' Visual Basic
Public Sub StartSynchronizationDelete()

// C#
public void StartSynchronizationDelete();
```

Remarks

Once this function is called, all delete operations are again synchronized, causing the rows deleted from the UltraLite database to be removed from the consolidated database as well.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULConnection class” on page 72
- “ULConnection members” on page 72
- “StopSynchronizationDelete method” on page 96
- “Truncate method” on page 322

StopSynchronizationDelete method

UL Ext.: Prevents delete operations from being synchronized.

Prototypes

' Visual Basic
Public Sub **StopSynchronizationDelete()**

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

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULConnection class” on page 72
- “ULConnection members” on page 72
- “StartSynchronizationDelete method” on page 95

Synchronize method

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

Prototypes

' Visual Basic
Overloads Public Sub **Synchronize()**

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

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULConnection class” on page 72
- “ULConnection members” on page 72
- “Synchronize method” on page 96

Synchronize method

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

Prototypes

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

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

Parameters

- **listener** The object that will receive synchronization progress events.

Remarks

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

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

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

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULConnection class" on page 72
- "ULConnection members" on page 72
- "ULSyncProgressListener interface" on page 281
- "Synchronize method" on page 96

InfoMessage event

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

Prototypes

```
' Visual Basic
Public Event InfoMessage As ULInfoMessageEventHandler

// C#
public event ULInfoMessageEventHandler InfoMessage;
```

Remarks

To process UltraLite.NET warnings or informational messages, you must create a ULInfoMessageEventHandler delegate delegate and attach it to this event.

Example

The following code defines an informational message event handler.

```
' Visual Basic
Private Sub MyInfoMessageHandler( _
      obj As Object, args As ULInfoMessageEventArgs _
    )
  System.Console.WriteLine( _
      "InfoMesageHandler: " + args.NativeError + ", " _
      + args.Message _
    )
End Sub

// C#
private void MyInfoMessageHandler(
      object obj, ULInfoMessageEventArgs args
    )
{
  System.Console.WriteLine(
      "InfoMesageHandler: " + args.NativeError + ", "
      + args.Message
    );
}
```

The following code adds the MyInfoMessageHandler to the connection named conn.

```
' Visual Basic
AddHandler conn.InfoMessage, AddressOf MyInfoMessageHandler

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

StateChange event

Occurs when this connection changes state.

Prototypes

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

// C#
public event StateChangeEventHandler **StateChange**;

Remarks

To process state change messages, you must create a StateChangeEventHandler delegate and attach it to this event.

Example

The following code defines a state change event handler.

```
' Visual Basic
Private Sub MyStateHandler( _
      obj As Object, args As StateChangeEventArgs _
    )
  System.Console.WriteLine( _
      "StateHandler: " + args.OriginalState + " to " _
      + args.CurrentState _
    )
End Sub

// C#
private void MyStateHandler(
      object obj, StateChangeEventArgs args
    )
{
  System.Console.WriteLine(
      "StateHandler: " + args.OriginalState + " to "
      + args.CurrentState
    );
}
```

The following code adds the MyStateHandler to the connection named conn.

```
' Visual Basic
AddHandler conn.StateChange, AddressOf MyStateHandler

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

ULConnectionParms class

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

Prototypes

```
' Visual Basic
Public Class ULConnectionParms
  Inherits Component
```

```
// C#
public class ULConnectionParms :
  Component
```

Remarks

A ULConnectionParms object is used to specify the parameters for opening a connection (Open method), creating a database (OpenWithCreate method), dropping a database (DropDatabase method), or specifying a schema file (ApplyFile method).

Leading and trailing spaces are ignored and values must not contain semicolons (;), parentheses (), or curly braces {}. For future compatibility it is strongly recommended that values do not contain single (‘) or double (“) quotes.

When building a connection string, you need to identify the database, specify any optional connection settings, and specify the database schema file if you wish to create the database with OpenWithCreate method. Once you have supplied all the connection parameters by setting the appropriate properties on a ULConnectionParms object, you create a connection string using the ToString method method. The resulting string is used to create a new ULConnection class with the ULConnection constructor constructor or set the ConnectionString property property of an existing ULConnection class object.

Identifying the database

Each instance contains platform-specific paths to the database. Only the value corresponding to the executing platform is used. For example, in the code below the path \UltraLite\mydb1.udb would be used on Windows CE, while mydb2.db would be used on other platforms.

```
' Visual Basic
Dim dbName As ULConnectionParms = new ULConnectionParms
dbName.DatabaseOnCE = "\UltraLite\mydb1.udb"
dbName.DatabaseOnDesktop = "mydb2.udb"

// C#
ULConnectionParms dbName = new ULConnectionParms();
dbName.DatabaseOnCE = "\\UltraLite\\mydb1.udb";
dbName.DatabaseOnDesktop = "mydb2.udb";
```

The recommended extension for UltraLite database files is .udb. On Windows CE devices, the default database is \UltraLiteDB\ulstore.udb. On other Windows platforms, the default database is ulstore.udb. In C#, you must escape any backslash characters in paths or use @-quoted string literals.

If you are using multiple databases, you should specify a database name for each database. See the AdditionalParms property property for more information.

Optional connection settings

Depending on your application's needs and how the database was created, you may need to supply a non-default UserID property and Password property, a database EncryptionKey property, and the connection CacheSize property. If your application is using multiple connections, you should provide a unique ConnectionName property for each connection.

Databases are created with a single authenticated user, DBA, whose initial password is SQL. By default, connections are opened using the user ID DBA and password SQL. To disable the default user, use the RevokeConnectFrom method method. To add a user or change a user's password, use the GrantConnectTo method method.

If an encryption key was supplied when the database was created, all subsequent connections to the database must use the same encryption key. To change a database's encryption key, use the ChangeEncryptionKey method method.

Identifying the schema

When creating a database, you must supply a schema. The path to the schema file is specified using the platform-specific properties SchemaOnCE property and SchemaOnDesktop property. Only the value corresponding to the executing platform is used.

For more information, see "Connection Parameters" [*UltraLite Database User's Guide*, page 63].

ULConnectionParms members

Public instance constructors

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

Public instance properties

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

Public instance methods

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

Public instance events

Member	Description
UnusedEvent event	Unused.

ULConnectionParms constructor

Initializes a ULConnectionParms instance with its default values.

Prototypes

```
' Visual Basic
Public Sub New()
```

```
// C#
public ULConnectionParms();
```

AdditionalParms property

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

Prototypes

```
' Visual Basic
Public Property AdditionalParms As String
```

```
// C#
public string  AdditionalParms {get;set;}
```

Property value

A semicolon-separated list of keyword=value additional parameters. The default is a null reference (Nothing in Visual Basic).

Remarks

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

Additional parameters are:

Keyword	Description
dbn	Identifies a loaded database to which a connection needs to be made. When a database is started, it is assigned a database name, either explicitly with the dbn parameter, or by UltraLite using the base of the filename with the extension and path removed. When opening connections, UltraLite first searches for a running database with a matching dbn. If one is not found, UltraLite starts a new database using the appropriate database filename parameter (DatabaseOnCE property or DatabaseOnDesktop property). This parameter is required if the application (or UltraLite engine) needs to access two different databases that have the same base filename. This parameter is only used when opening a connection with Open method or OpenWithCreate method.

Keyword	Description
obfuscate	When set to one (1), specifies that the database is to be created with obfuscation. For example: `createParms.AdditionalParms = "obfuscate=1"` Obfuscation provides security against straightforward attempts to view data in the database directly using a viewing tool. It is not proof against skilled and determined attempts to gain access to the data. Obfuscation has little or no performance impact. This parameter is only used when creating a new database with OpenWithCreate method.
page_size	UltraLite databases are stored in pages. I/O operations are carried out one page at a time. The default page size for UltraLite databases is 4 KB. You can specify 2 KB pages using the following storage parameters string: `connParms.AdditionalParms = "page_size=2k"` Setting a page size of 2 KB reduces the maximum number of tables to approximately 500. This parameter is only used when creating a new database with OpenWithCreate method.

Keyword	Description
reserve_size	Reserves file system space for storage of UltraLite persistent data. The reserve_size parameter allows you to pre-allocate the file system space required for your UltraLite database without inserting any data. Reserving file system space can improve performance slightly and also prevent out of memory failures. By default, the persistent storage file only grows when required as the application updates the database. Note that reserve_size reserves file system space, which includes the metadata in the persistent store file, and not just the raw data. The metadata overhead, as well as data compression, must be considered when deriving the required file system space from the amount of database data. Running the database with test data and observing the persistent store file size is recommended. The reserve_size parameter reserves space by growing the persistent store file to the given reserve size on startup, regardless of whether the file previously existed. The file is never truncated. The following parameter string ensures that the persistent store file is at least 2 MB upon startup. `createParms.AdditionalParms = "reserve_size=2m"` This parameter is only used when opening a connection with Open method or OpenWithCreate method.

Exceptions

- ArgumentException - The value contained parentheses () or curly braces {}.

CacheSize property

Specifies the size of the cache.

Prototypes

' Visual Basic
Public Property **CacheSize** As String

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

Property value

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

Remarks

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

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

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

The default cache size is sixteen 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 may interfere with the number of other applications you can use.

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

Exceptions

- ArgumentException - The value contained a semi-colon (;), parentheses () or curly braces {}.

ConnectionName property

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

Prototypes

' Visual Basic
Public Property **ConnectionName** As String

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

Property value

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

Exceptions

- ArgumentException - The value contained a semi-colon (;), parentheses () or curly braces {}.

DatabaseOnCE property

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

Prototypes

' Visual Basic
Public Property **DatabaseOnCE** As String

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

Property value

A string specifying the full path to the database. If the value is a null reference (Nothing in Visual Basic), the default database \UltraLiteDB\ulstore.udb is used. In C#, you must escape any backslash characters in paths or use @-quoted string literals.

Exceptions

- ArgumentException - The value contained a semicolon (;), parentheses (), or curly braces {}.

DatabaseOnDesktop property

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

Prototypes

```
' Visual Basic
Public Property DatabaseOnDesktop As String

// C#
public string  DatabaseOnDesktop {get;set;}
```

Property value

A string specifying the absolute or relative path to the database. If the value is a null reference (Nothing in Visual Basic), the default database ulstore.udb is used. In C#, you must escape any backslash characters in paths or use @-quoted string literals.

Exceptions

- ArgumentException - The value contained a semi-colon (;), parentheses () or curly braces {}.

EncryptionKey property

Specifies a key for encrypting the database.

Prototypes

```
' Visual Basic
Public Property EncryptionKey As String

// C#
public string  EncryptionKey {get;set;}
```

Property value

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

Remarks

All connections must use the same key as was specified when the database was created. Lost or forgotten keys result in completely inaccessible databases.

As with all passwords, it is best to choose a key value that cannot be easily guessed. The key can be of arbitrary length, but generally the longer the key, the better, because a shorter key is easier to guess than a longer one. As well, including a combination of numbers, letters, and special characters decreases the chances of someone guessing the key.

Exceptions

- ArgumentException - The value contained a semi-colon (;), parentheses () or curly braces {}.

See also

- "ULConnectionParms class" on page 100
- "ULConnectionParms members" on page 101
- "ChangeEncryptionKey method" on page 84

Password property

Specifies the password for the authenticated user.

Prototypes

```
' Visual Basic
Public Property Password As String

// C#
public string  Password {get;set;}
```

Property value

A string specifying a database user ID. The default is SQL.

Remarks

When a database is created, the password for the DBA user ID is set to SQL. Passwords are case-insensitive if the database is case-insensitive and case-sensitive if the database is case-sensitive.

Exceptions

- ArgumentException - The value contained a semi-colon (;), parentheses () or curly braces {}.

See also

- "ULConnectionParms class" on page 100
- "ULConnectionParms members" on page 101
- "UserID property" on page 109

SchemaOnCE property

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

Prototypes

```
' Visual Basic
Public Property SchemaOnCE As String

// C#
public string  SchemaOnCE {get;set;}
```

Property value

A string specifying the full path to the schema file. The default is a null reference (Nothing in Visual Basic) meaning no schema file is specified for this platform.

Exceptions

- ArgumentException - The value contained a semi-colon (;), parentheses () or curly braces {}.

SchemaOnDesktop property

Specifies the path and filename of the UltraLite schema on Windows desktop

platforms.

Prototypes

' Visual Basic
Public Property **SchemaOnDesktop** As String

// C#
public string **SchemaOnDesktop** {get;set;}

Property value

A string specifying the absolute or relative path to the schema file. The default is a null reference (Nothing in Visual Basic) meaning no schema file is specified for this platform.

Exceptions

- ArgumentException - The value contained a semi-colon (;), parentheses () or curly braces {}.

UserID property

Specifies an authenticated user for the database.

Prototypes

' Visual Basic
Public Property **UserID** As String

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

Property value

A string specifying a database user ID. The default value is DBA.

Remarks

Databases are initially created with a single authenticated user named DBA. User IDs are case-insensitive if the database is case-insensitive and case-sensitive if the database is case-sensitive.

Exceptions

- ArgumentException - The value contained a semi-colon (;), parentheses () or curly-braces {}.

See also

- "ULConnectionParms class" on page 100
- "ULConnectionParms members" on page 101
- "Password property" on page 108
- "GrantConnectTo method" on page 92
- "RevokeConnectFrom method" on page 94
- "IsCaseSensitive property" on page 143

ToString method

Returns the string representation of this instance.

Prototypes

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

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

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

UnusedEvent event

Unused.

Prototypes

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

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

Remarks

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

ULConnectionParms.UnusedEventHandler delegate

UL Ext.: Unused.

Prototypes

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

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

Parameters

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

Remarks

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

ULCursorSchema class

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

Prototypes

' Visual Basic
MustInherit Public Class **ULCursorSchema**

// C#
public abstract class **ULCursorSchema**

Remarks

This class is an abstract base class of the ULTableSchema class and ULResultSetSchema class classes.

Note to users porting from the iAnywhere.UltraLite namespace: Column IDs are 0-based, not 1-based as they are in the iAnywhere.UltraLite namespace.

ULCursorSchema members

Public instance properties

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

Public instance methods

Member	Description
GetColumnID method	Returns the column ID of the named column.
GetColumnName method	Returns the name of the column identified by the specified column ID.
GetColumnPrecision method	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
GetColumnScale method	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
GetColumnSize method	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
GetColumnULDbType method	Returns the UltraLite.NET data type of the column identified by the specified column ID.
GetSchemaTable method	Returns a DataTable that describes the column schema of the ULDataReader class.

Protected instance methods

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

ColumnCount property

Returns the number of columns in the cursor.

Prototypes

' Visual Basic
Public Readonly Property **ColumnCount** As Short

// C#
public short **ColumnCount** {get;}

Property value

The number of columns in the cursor or 0 if the cursor schema is closed.

Remarks

Column IDs range from 0 to ColumnCount-1, inclusively.

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.

IsOpen property

Checks whether the cursor schema is currently open.

Prototypes

' Visual Basic
Public Readonly Property **IsOpen** As Boolean

// C#
public bool **IsOpen** {get;}

Property value

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

Name property

Returns the name of the cursor.

Prototypes

' Visual Basic
Public Readonly Property **Name** As String

// C#
public string **Name** {get;}

Property value

The name of the cursor as a string.

Finalize method

Releases the unmanaged resources used by the ULCursorSchema.

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()

// C#
protected override void Finalize();
```

GetColumnID method

Returns the column ID of the named column.

Prototypes

```
' Visual Basic
Public Function GetColumnID( _
  ByVal name As String _
) As Short

// C#
public short GetColumnID(
  string  name
);
```

Parameters

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

Return value

The column ID of the named column.

Remarks

Column IDs range from 0 to ColumnCount property-1, inclusively.

Note that in result sets, not all columns have names and not all column names are unique. If you are not using aliases, the name of a non-computed column is prefixed with the name of the table the column is from. For example, MyTable.ID is the name of the only column in the result set for the query “SELECT ID FROM MyTable”.

Column IDs and counts may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULCursorSchema class” on page 112
- “ULCursorSchema members” on page 112
- “ColumnCount property” on page 113

GetColumnName method

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

Prototypes

```
' Visual Basic
Public Function GetColumnName( _
  ByVal columnID As Integer _
) As String

// C#
public string  GetColumnName(
  int columnID
);
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount property-1].

Return value

The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned.

Remarks

Note that in result sets, not all columns have names and not all column names are unique. If you are not using aliases, the name of a non-computed column is prefixed with the name of the table the column is from. For example, MyTable.ID is the name of the only column in the result set for the query “SELECT ID FROM MyTable”.

Column IDs and count may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULCursorSchema class” on page 112
- “ULCursorSchema members” on page 112
- “ColumnCount property” on page 113

GetColumnPrecision method

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

Prototypes

```
' Visual Basic
Public Function GetColumnPrecision( _
  ByVal columnID As Integer _
) As Integer

// C#
public int GetColumnPrecision(
  int columnID
);
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount property-1].

Return value

The precision of the specified numeric column.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULCursorSchema class” on page 112
- “ULCursorSchema members” on page 112
- “GetColumnULDbType method” on page 117

GetColumnScale method

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

Prototypes

```
' Visual Basic
Public Function GetColumnScale( _
  ByVal columnID As Integer _
) As Integer
```

```
// C#
public int GetColumnScale(
  int columnID
);
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount property-1].

Return value

The scale of the specified numeric column.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULCursorSchema class” on page 112
- “ULCursorSchema members” on page 112
- “GetColumnULDbType method” on page 117

GetColumnSize method

Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).

Prototypes

```
' Visual Basic
Public Function GetColumnSize( _
  ByVal columnID As Integer _
) As Integer
```

```
// C#
public int GetColumnSize(
  int columnID
);
```

Parameters

- **columnID** ID of the column. The value must be in the range [0,ColumnCount property-1].

Return value — The size of the specified sized column.

Exceptions
- ULException class - A SQL error occurred.

See also
- "ULCursorSchema class" on page 112
- "ULCursorSchema members" on page 112
- "GetColumnULDbType method" on page 117

GetColumnULDbType method

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

Prototypes

```
' Visual Basic
Public Function GetColumnULDbType( _
  ByVal columnID As Integer _
) As ULDbType
```

```
// C#
public ULDbType GetColumnULDbType(
  int columnID
);
```

Parameters
- **columnID** ID of the column. The value must be in the range [0,ColumnCount property-1].

Return value — A ULDbType enumerated integer.

Exceptions
- ULException class - A SQL error occurred.

See also
- "ULCursorSchema class" on page 112
- "ULCursorSchema members" on page 112
- "ULDbType enumeration" on page 182
- "ColumnCount property" on page 113

GetSchemaTable method

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

Prototypes

```
' Visual Basic
Public Function GetSchemaTable() As DataTable
```

```
// C#
public DataTable GetSchemaTable();
```

Return value — A DataTable that describes the column schema.

Remarks — For more information see GetSchemaTable method.

ULDataAdapter class

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

Prototypes

```
' Visual Basic
NotInheritable Public Class ULDataAdapter
  Implements IDbDataAdapter, IDataAdapter, IDisposable
```

```
// C#
public sealed class ULDataAdapter :
  IDbDataAdapter, IDataAdapter,
  IDisposable
```

Remarks

The DataSet provides a way to work with data offline; that is, away from your UltraLite database. The ULDataAdapter provides methods to associate a DataSet with a set of SQL statements.

Since UltraLite is a local database and MobiLink has conflict resolution, the use of the ULDataAdapter is limited. For most purposes, the ULDataReader class or ULTable class classes provide more efficient access to data.

Inherits: DbDataAdapter

Implements: IDbDataAdapter, IDataAdapter, IDisposable

ULDataAdapter members

Public instance constructors

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

Public instance properties

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

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

Public instance methods

Member	Description
Dispose method	Releases the unmanaged resources used by the ULDataAdapter and optionally releases the managed resources.
Fill method	Adds or refreshes rows into the DataTable named “Table” of the specified DataSet. The DataTable named “Table” is created and added to the DataSet if necessary.
Fill method	Adds or refreshes rows into the named DataTable of the specified DataSet. The DataTable named is created and added to the DataSet if neccessary.
Fill method	Adds or refreshes the specified range of rows into the named DataTable of the specified DataSet. The DataTable named is created and added to the DataSet if neccessary.

Member	Description
Fill method	Adds or refreshes rows into the specified DataTable.
FillSchema method	Adds a DataTable named “Table” to a DataSet and configures the schema to match the schema in the data source.
FillSchema method	Adds a DataTable named “Table” to a DataSet and configures the schema to match the schema in the data source.
FillSchema method	Configures the schema of a DataTable to match the schema in the data source.
GetFillParameters method	Returns the parameters set by the user when executing a SELECT statement.
Update method	Updates the rows in the database with the changes made to the DataTable named “Table” of the specified DataSet.
Update method	Updates the rows in the database with the changes made to the named DataTable of the specified DataSet.
Update method	Updates the rows in the database with the changes made to the specified DataTable.
Update method	Updates the rows in the database with the changes made to the specified DataRow array.

Public instance events

Member	Description
FillError event	Occurs when an error is detected during a fill operation.
RowUpdated event	Occurs during an update after a command is executed against the data source. When an attempt to update is made, the event fires.
RowUpdating event	Occurs during an update before a command is executed against the data source. When an attempt to update is made, the event fires.

ULDataAdapter constructor

Initializes a ULDataAdapter object.

Prototypes

’ Visual Basic
Overloads Public Sub **New()**

// C#
public **ULDataAdapter();**

See also

- "ULDataAdapter class" on page 118
- "ULDataAdapter members" on page 118
- "ULDataAdapter constructor" on page 121
- "ULDataAdapter constructor" on page 121
- "ULDataAdapter constructor" on page 122

ULDataAdapter constructor

Initializes a ULDataAdapter object with the specified SELECT statement.

Prototypes

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

// C#
public ULDataAdapter(
  ULCommand selectCommand
);
```

Parameters

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

See also

- "ULDataAdapter class" on page 118
- "ULDataAdapter members" on page 118
- "ULDataAdapter constructor" on page 120
- "ULDataAdapter constructor" on page 121
- "ULDataAdapter constructor" on page 122

ULDataAdapter constructor

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

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal selectCommandText As String, _
  ByVal selectConnection As ULConnection _
)

// C#
public ULDataAdapter(
  string selectCommandText,
  ULConnection selectConnection
);
```

Parameters

- **selectCommandText** A SELECT statement to be used by the SelectCommand property property of the ULDataAdapter.

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

See also

- "ULDataAdapter class" on page 118
- "ULDataAdapter members" on page 118
- "ULDataAdapter constructor" on page 120
- "ULDataAdapter constructor" on page 121
- "ULDataAdapter constructor" on page 122

ULDataAdapter constructor

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

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal selectCommandText As String, _
  ByVal selectConnectionString As String _
)

// C#
public ULDataAdapter(
  string selectCommandText,
  string selectConnectionString
);
```

Parameters

- **selectCommandText** A SELECT statement to be used by the SelectCommand property property of the ULDataAdapter.
- **selectConnectionString** A connection string for an UltraLite.NET database.

See also

- "ULDataAdapter class" on page 118
- "ULDataAdapter members" on page 118
- "ULDataAdapter constructor" on page 120
- "ULDataAdapter constructor" on page 121
- "ULDataAdapter constructor" on page 121

AcceptChangesDuringFill property

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

Prototypes

```
' Visual Basic
Public Property AcceptChangesDuringFill As Boolean

// C#
public bool AcceptChangesDuringFill {get;set;}
```

Property value

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

ContinueUpdateOnError property

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

Prototypes

' Visual Basic
Public Property **ContinueUpdateOnError** As Boolean

// C#
public bool **ContinueUpdateOnError** {get;set;}

Property value

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

Remarks

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

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

DeleteCommand property

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

Prototypes

' Visual Basic
Public Property **DeleteCommand** As ULCommand

// C#
public ULCommand **DeleteCommand** {get;set;}

Property value

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

Remarks

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

This is the strongly typed version of IDbDataAdapter.DeleteCommand.

InsertCommand property

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

Prototypes

```
' Visual Basic
Public Property InsertCommand As ULCommand
```

```
// C#
public ULCommand InsertCommand {get;set;}
```

Property value

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

Remarks

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

This is the strongly typed version of IDbDataAdapter.InsertCommand.

MissingMappingAction property

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

Prototypes

```
' Visual Basic
NotOverridable Public Property MissingMappingAction As MissingMappingAction
    _
  Implements IDataAdapter.MissingMappingAction
```

```
// C#
public MissingMappingAction MissingMappingAction {get;set;}
```

Property value

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

Implements

IDataAdapter.MissingMappingAction

MissingSchemaAction property

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

Prototypes

```
' Visual Basic
NotOverridable Public Property MissingSchemaAction As MissingSchemaAction _
  Implements IDataAdapter.MissingSchemaAction
```

// C#
public MissingSchemaAction **MissingSchemaAction** {get;set;}

Property value — One of the MissingSchemaAction values. The default is MissingSchemaAction.Add.

Implements — IDataAdapter.MissingSchemaAction

SelectCommand property

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

Prototypes

' Visual Basic
Public Property **SelectCommand** As ULCommand

// C#
public ULCommand **SelectCommand** {get;set;}

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

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

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

This is the strongly typed version of IDbDataAdapter.SelectCommand.

TableMappings property

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

Prototypes

' Visual Basic
Public Readonly Property **TableMappings** As DataTableMappingCollection

// C#
public DataTableMappingCollection **TableMappings** {get;}

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

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

This is the strongly typed version of IDataAdapter.TableMappings.

UpdateCommand property

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

Prototypes

```
' Visual Basic
Public Property UpdateCommand As ULCommand

// C#
public ULCommand UpdateCommand {get;set;}
```

Property value

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

Remarks

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

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

This is the strongly typed version of IDbDataAdapter.UpdateCommand.

Dispose method

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

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Sub Dispose() _
  Implements IDisposable.Dispose

// C#
public void Dispose();
```

Implements

IDisposable.Dispose

Fill method

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

Prototypes

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

// C#
public int Fill(
  DataSet dataSet
);
```

Parameters

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

Return value

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

Remarks

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

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

For more information, see IDataAdapter.Fill.

Exceptions

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

Implements

IDataAdapter.Fill

Fill method

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

Prototypes

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

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

Parameters

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

Return value

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

Remarks

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

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

For more information, see IDataAdapter.Fill.

Exceptions

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

Fill method

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

Prototypes

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

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

Parameters

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

Return value

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

Remarks

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

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

For more information, see IDataAdapter.Fill.

Exceptions

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

Fill method

Adds or refreshes rows into the specified DataTable.

Prototypes

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

// C#
public int Fill(
  DataTable dataTable
);
```

Parameters

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

Return value

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

Remarks

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

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

For more information, see IDataAdapter.Fill.

Exceptions

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

FillSchema method

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

Prototypes

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

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

Parameters

- **dataSet** A DataSet to fill with the schema.

- **schemaType** One of the SchemaType values that specify how to insert the schema.

Return value

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

Remarks

For more information, see IDataAdapter.FillSchema.

Exceptions

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

Implements

IDataAdapter.FillSchema

FillSchema method

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

Prototypes

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

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

Parameters

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

Return value

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

Remarks

For more information, see DbDataAdapter.FillSchema.

Exceptions

- System.ArgumentNullException - The *dataSet* or *srcTable* parameter is invalid.

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

FillSchema method

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

Prototypes

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

```
// C#
public DataTable FillSchema(
  DataTable dataTable,
  SchemaType schemaType
);
```

Parameters

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

Return value

A reference to the DataTable object that contains the schema.

Remarks

For more information, see DbDataAdapter.FillSchema.

Exceptions

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

GetFillParameters method

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

Prototypes

```
' Visual Basic
Public Function GetFillParameters() As ULParameter()
```

```
// C#
public ULParameter[] GetFillParameters();
```

Return value

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

Update method

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

Prototypes

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

// C#
public int Update(
  DataSet dataSet
);
```

Parameters

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

Return value

The number of rows successfully updated from the DataSet.

Remarks

For more information, see IDataAdapter.Update.

Exceptions

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

Implements

IDataAdapter.Update

Update method

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

Prototypes

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

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

Parameters

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

Return value

The number of rows successfully updated from the DataSet.

Remarks

For more information, see DbDataAdapter.Update.

Exceptions

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

Update method

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

Prototypes

```
' Visual Basic
Overloads Public Function Update( _
  ByVal dataTable As DataTable _
) As Integer

// C#
public int Update(
  DataTable dataTable
);
```

Parameters

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

Return value

The number of rows successfully updated from the DataTable.

Remarks

For more information, see DbDataAdapter.Update.

Exceptions

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

Update method

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

Prototypes

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

// C#
public int Update(
  DataRow[] dataRows
);
```

Parameters

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

Return value
The number of rows successfully updated from the DataRow array.

Remarks
For more information, see DbDataAdapter.Update.

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

FillError event

Occurs when an error is detected during a fill operation.

Prototypes

' Visual Basic
Public Event **FillError** As FillErrorEventHandler

// C#
public event FillErrorEventHandler **FillError**;

Remarks
The FillError event allows you to determine whether or not the fill operation should continue after the error occurs.

Examples of when the FillError event might occur are:

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

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

RowUpdated event

Occurs during an update after a command is executed against the data source. When an attempt to update is made, the event fires.

Prototypes

' Visual Basic
Public Event **RowUpdated** As ULRowUpdatedEventHandler

// C#
public event ULRowUpdatedEventHandler **RowUpdated**;

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

RowUpdating event

Occurs during an update before a command is executed against the data source. When an attempt to update is made, the event fires.

Prototypes

```
' Visual Basic
Public Event RowUpdating As ULRowUpdatingEventHandler

// C#
public event ULRowUpdatingEventHandler RowUpdating;
```

Remarks

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

ULDatabaseManager class

UL Ext.: Manages synchronization listeners and the UltraLite.NET runtime type. The ULDatabaseManager class also allows you to drop (delete) UltraLite.NET databases.

Prototypes

' Visual Basic
NotInheritable Public Class **ULDatabaseManager**

// C#
public sealed class **ULDatabaseManager**

Remarks

This class is a singleton class whose only instance is accessible only through the static (Shared in Visual Basic) DatabaseManager property property.

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

ULDatabaseManager members

Public static properties (Shared)

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

Public instance methods

Member	Description
DropDatabase method	Deletes the specified database. You cannot drop a database that has open connections.
SetActiveSyncListener method	Specifies the listener object used to process ActiveSync calls from the MobiLink provider for ActiveSync.
SetServerSyncListener method	Specifies the listener object used to process the specified server synchronization message.

Protected instance methods

Member	Description
Finalize method	Releases the unmanaged resources used by the ULDatabaseManager.

RuntimeType property

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

Prototypes

' Visual Basic
Public Shared Property **RuntimeType** As ULRuntimeType

// C#
public const ULRuntimeType **RuntimeType** {get;set;}

Property value

A ULRuntimeType enumeration value identifying the type of the unmanaged UltraLite.NET runtime.

Example

The following example selects the UltraLite Engine runtime and creates a connection.

```
' Visual Basic
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT
Dim conn As ULConnection = new ULConnection
' The RuntimeType is now locked

// C#
ULDatabaseManager.RuntimeType = ULRuntimeType.UL_ENGINE_CLIENT;
ULConnection conn = new ULConnection();
// The RuntimeType is now locked
```

DropDatabase method

Deletes the specified database.

You cannot drop a database that has open connections.

Prototypes

' Visual Basic
Public Sub **DropDatabase(** _
  ByVal *parms* As ULConnectionParms _
)

// C#
public void **DropDatabase(**
  ULConnectionParms *parms*
);

Parameters

- **parms** The parameters for identifying a database. See ULConnectionParms class for more information.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDatabaseManager class" on page 137
- "ULDatabaseManager members" on page 137

- "OpenWithCreate method" on page 93

Finalize method

Releases the unmanaged resources used by the ULDatabaseManager.

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()

// C#
protected override void Finalize();
```

SetActiveSyncListener method

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

Prototypes

```
' Visual Basic
Public Sub SetActiveSyncListener( _
  ByVal appClassName As String, _
  ByVal listener As ULActiveSyncListener _
)

// C#
public void SetActiveSyncListener(
  string appClassName,
  ULActiveSyncListener listener
);
```

Parameters

- **appClassName** The unique class name for the application. This is the class name used when the application is registered for use with ActiveSync.
- **listener** The ULActiveSyncListener interface object. Use null (Nothing in Visual Basic) to remove the previous listener.

Remarks

The parameter *appClassName* is the unique identifier used to identify the application. The application can only use one *appClassName* at a time. While a listener is registered with a particular *appClassName*, calls to SetServerSyncListener method or SetActiveSyncListener method with a different *appClassName* fail.

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

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

Applications should remove all listeners prior to exiting.

Exceptions

- ULException class - A SQL error occurred.

Example

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

See also

- "ULDatabaseManager class" on page 137
- "ULDatabaseManager members" on page 137
- "ColumnCount property" on page 113

SetServerSyncListener method

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

Prototypes

```
' Visual Basic
Public Sub SetServerSyncListener( _
  ByVal messageName As String, _
  ByVal appClassName As String, _
  ByVal listener As ULServerSyncListener _
)

// C#
public void SetServerSyncListener(
  string messageName,
  string appClassName,
  ULServerSyncListener listener
);
```

Parameters

- **messageName** The name of the message.
- **appClassName** The unique class name for the application. This is a unique identifier used to identify the application.
- **listener** The ULServerSyncListener interface object. Use null (Nothing in Visual Basic) to remove the previous listener.

Remarks

The parameter *appClassName* is the unique identifier used to identify the application. The application may only use one *appClassName* at a time. While a listener is registered with a particular *appClassName*, calls to SetServerSyncListener method or SetActiveSyncListener method with a different *appClassName* fail.

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

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

Applications should remove all listeners before exiting.

Exceptions

- ULException class - A SQL error occurred.

Example

Refer to the ServerSyncInvoked method documentation for an example of SetServerSyncListener method.

ULDatabaseSchema class

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

Prototypes

' Visual Basic
NotInheritable Public Class **ULDatabaseSchema**

// C#
public sealed class **ULDatabaseSchema**

Remarks

This class cannot be directly instantiated. A ULDatabaseSchema class object is attached to a connection as its Schema property property and is only valid while that connection is open.

ULDatabaseSchema members

Public instance properties

Member	Description
CollationName property	The name of the database's collation sequence.
IsCaseSensitive property	Checks whether the database is case sensitive.
IsOpen property	Whether or not the database schema is open.
PublicationCount property	The number of publications in the database.
Signature property	The signature of this database.
TableCount property	The number of tables in the database.

Public instance methods

Member	Description
ApplyFile method	Applies a database schema file to the database while allowing a listener to monitor the progress.
ApplyFile method	Applies a database schema file to the database.
GetDatabaseProperty method	Returns the value of the specified database property.
GetPublicationName method	Returns the name of the publication identified by the specified publication ID. Publication IDs are not publication masks.
GetPublicationSchema method	Returns the publication schema corresponding to the named publication.
GetTableCountInPublications method	Returns the number of tables included in the specified publication mask.

Member	Description
GetTableName method	Returns the name of the table identified by the specified table ID.

CollationName property

The name of the database's collation sequence.

Prototypes

' Visual Basic
Public Readonly Property **CollationName** As String

// C#
public string **CollationName** {get;}

Property value

A string representing the database's collation sequence.

Remarks

The database collation sequence affects how indexes on tables and result sets are sorted.

Exceptions

- ULException class - A SQL error occurred.

IsCaseSensitive property

Checks whether the database is case sensitive.

Prototypes

' Visual Basic
Public Readonly Property **IsCaseSensitive** As Boolean

// C#
public bool **IsCaseSensitive** {get;}

Property value

True if the database is case sensitive, and false if the database is case insensitive.

Remarks

Database case sensitivity affects how indexes on tables and result sets are sorted. Case sensitivity also affects how UserID property and Password property are verified.

Exceptions

- ULException class - A SQL error occurred.

IsOpen property

Whether or not the database schema is open.

Prototypes

' Visual Basic
Public Readonly Property **IsOpen** As Boolean

// C#
public bool **IsOpen** {get;}

Property value — True if this database schema is currently open, false if this database schema is currently closed.

Remarks — A ULDatabaseSchema object is open only if the connection it is attached to is open.

PublicationCount property

The number of publications in the database.

Prototypes

```
' Visual Basic
Public Readonly Property PublicationCount As Integer
```

```
// C#
public int PublicationCount {get;}
```

Property value — The numberof publications in the database or zero if the connection is not open.

Remarks — Publication IDs range from 1 to PublicationCount, inclusively. Publication IDs are not publication masks.

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

See also
- “ULDatabaseSchema class” on page 142
- “ULDatabaseSchema members” on page 142
- “GetPublicationName method” on page 147

Signature property

The signature of this database.

Prototypes

```
' Visual Basic
Public Readonly Property Signature As String
```

```
// C#
public string  Signature {get;}
```

Property value — A string representing the signature of the currently opened database.

Exceptions
- ULException class - A SQL error occurred.

TableCount property

The number of tables in the database.

Prototypes

```
' Visual Basic
Public Readonly Property TableCount As Integer
```

```
// C#
public int TableCount {get;}
```

Property value — The number of tables in the database or zero if the connection is not open.

Remarks — Table IDs range from 1 to TableCount, inclusively.

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

ApplyFile method

Applies a database schema file to the database while allowing a listener to monitor the progress.

Prototypes

```
' Visual Basic
Overloads Public Sub ApplyFile( _
  ByVal parms As ULConnectionParms, _
  ByVal listener As ULSchemaUpgradeListener _
)
```

```
// C#
public void ApplyFile(
  ULConnectionParms parms,
  ULSchemaUpgradeListener listener
);
```

Parameters

- **parms** Parameters for specifying the schema to be applied to the database. See ULConnectionParms class for more information.
- **listener** The ULSchemaUpgradeListener interface object to receive schema upgrade progress events.

Remarks — All instances of ULResultSetSchema class, ULTableSchema class, ULIndexSchema class, and ULPublicationSchema class will be invalidated and will need to be replaced.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULDatabaseSchema class” on page 142
- “ULDatabaseSchema members” on page 142
- “ApplyFile method” on page 145
- “ULConnectionParms class” on page 100

ApplyFile method

Applies a database schema file to the database.

Prototypes

```
' Visual Basic
Overloads Public Sub ApplyFile( _
  ByVal parms As ULConnectionParms _
)

// C#
public void ApplyFile(
  ULConnectionParms parms
);
```

Parameters

- **parms** Parameters for specifying the schema to be applied to the database. See ULConnectionParms class for more information.

Remarks

All instances of ULResultSetSchema class, ULTableSchema class, ULIndexSchema class, and ULPublicationSchema class will be invalidated and will need to be replaced.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDatabaseSchema class" on page 142
- "ULDatabaseSchema members" on page 142
- "ApplyFile method" on page 145
- "ULConnectionParms class" on page 100

GetDatabaseProperty method

Returns the value of the specified database property.

Prototypes

```
' Visual Basic
Public Function GetDatabaseProperty( _
  ByVal name As String _
) As String

// C#
public string  GetDatabaseProperty(
  string  name
);
```

Parameters

- **name** The name of the database property to inquire about.

Return value

The value of the property as a string.

Remarks

Recognized properties are:

- "DATE_FORMAT" The date format used for string conversions by the database.

  This format is not necessarily the same as the one used by DateTime.

- "DATE_ORDER" The date order used for string conversions by the database.

- ”NEAREST_CENTURY” The nearest century used for string conversions by the database.
- ”PRECISION” The floating point precision used for string conversions by the database.
- ”SCALE” The minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION during string conversions by the database.
- ”TIME_FORMAT” The time format used for string conversions by the database.

  This format is not necessarily the same as the one used by TimeSpan.
- ”TIMESTAMP_FORMAT” The timestamp format used for string conversions by the database.

  This format is not necessarily the same as the one used by DateTime.
- ”TIMESTAMP_INCREMENT” The minimum difference between two unique timestamps, in nanoseconds (1,000,000th of a second).

Exceptions

- ULException class - A SQL error occurred.

GetPublicationName method

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

Prototypes

```
' Visual Basic
Public Function GetPublicationName( _
  ByVal pubID As Integer _
) As String

// C#
public string  GetPublicationName(
  int pubID
);
```

Parameters

- **pubID** The ID of the publication. The value must be in the range [1,PublicationCount property].

Return value

The publication name as a string.

Remarks

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

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULDatabaseSchema class” on page 142
- “ULDatabaseSchema members” on page 142
- “PublicationCount property” on page 144

GetPublicationSchema method

Returns the publication schema corresponding to the named publication.

Prototypes

```
' Visual Basic
Public Function GetPublicationSchema( _
  ByVal name As String _
) As ULPublicationSchema

// C#
public ULPublicationSchema GetPublicationSchema(
  string  name
);
```

Parameters

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

Return value

The ULPublicationSchema class object representing the named publication.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULDatabaseSchema class” on page 142
- “ULDatabaseSchema members” on page 142
- “GetPublicationName method” on page 147
- “ULPublicationSchema class” on page 226

GetTableCountInPublications method

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

Prototypes

```
' Visual Basic
Public Function GetTableCountInPublications( _
  ByVal mask As Integer _
) As Integer

// C#
public int GetTableCountInPublications(
  int mask
);
```

Parameters

- **mask** The set of publications to check. See ULPublicationSchema class for information on building publication masks.

Return value

The number of tables included in set of publications.

Remarks

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

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

Exceptions

- ULException class - A SQL error occurred.

GetTableName method

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

Prototypes

```
' Visual Basic
Public Function GetTableName( _
  ByVal tableID As Integer _
) As String

// C#
public string  GetTableName(
  int tableID
);
```

Parameters

- **tableID** The ID of the table. The value must be in range [1,TableCount].

Return value

The table name as a string.

Remarks

Table IDs may change during a schema upgrade. To correctly identify a table, access it by name or refresh the cached IDs after a schema upgrade.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULDatabaseSchema class” on page 142
- “ULDatabaseSchema members” on page 142
- “TableCount property” on page 144

ULDataReader class

Represents a read-only bi-directional cursor in an UltraLite database. Cursors are sets of rows from either a table or the result set from a query.

Prototypes

```
' Visual Basic
Public Class ULDataReader
  Inherits MarshalByRefObject
  Implements IDataReader, IDisposable, IDataRecord,
    IEnumerable
```

```
// C#
public class ULDataReader :
  MarshalByRefobject, IDataReader, IDisposable,
  IDataRecord, IEnumerable
```

Remarks

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

```
' Visual Basic
Dim cmd As ULCommand = new ULCommand( _
    "SELECT emp_id FROM employee", conn _
  )
Dim reader As ULDataReader = cmd.ExecuteReader()

// C#
ULCommand cmd = new ULCommand(
    "SELECT emp_id FROM employee", conn
  );
ULDataReader reader = cmd.ExecuteReader();
```

UL Ext.: The ADO.NET standard only requires forward-only motion through the result set, but ULDataReader is bi-directional. ULDataReader's Move methods provide you with full flexibility when moving through results.

ULDataReader is a read-only result set. If you need a more flexible object to manipulate results, use a ULDataAdapter class. The ULDataReader retrieves rows as needed, whereas the ULDataAdapter class must retrieve all rows of a result set before you can carry out any action on the object. For large result sets, this difference gives the ULDataReader a much faster response time.

UL Ext.: All columns of a ULDataReader may be retrieved using GetString method.

Implements: IDataReader, IDataRecord, IDisposable

ULDataReader members

Public instance properties

Member	Description
Depth property	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
FieldCount property	Returns the number of columns in the cursor.
IsBOF property	**UL Ext.:** Checks whether the current row position is before the first row or not.
IsClosed property	Checks whether the cursor is currently open.
IsEOF property	**UL Ext.:** Checks whether the current row position is after the last row or not.
Item property	Returns the value of the specified column in its native format. In C#, this property is the indexer for the ULDataReader class.
Item property	Returns the value of the specified named column in its native format. In C#, this property is the indexer for the ULDataReader class.
RecordsAffected property	Returns the number of rows changed, inserted, or deleted by execution of the SQL statement. For SELECT statements or CommandType.-TableDirect tables, this value is -1.
RowCount property	**UL Ext.:** Returns the number of rows in the cursor.
Schema property	**UL Ext.:** Holds the schema of this cursor.

Public instance methods

Member	Description
Close method	Closes the cursor.
Dispose method	Releases the unmanaged resources used by the ULDataReader and optionally releases the managed resources.
GetBoolean method	Returns the value for the specified column as a Boolean.
GetByte method	Returns the value for the specified column as an unsigned 8-bit value (Byte).
GetBytes method	Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination Byte array.
GetBytes method	**UL Ext.:** Returns the value for the specified column as an array of Bytes. Only valid for columns of type ULDbType enumeration, ULDbType enumeration, or ULDbType enumeration.

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

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

Protected instance methods

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

Depth property

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

Prototypes

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

// C#
```
public int Depth {get;}
```

Property value

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

Exceptions

- ULException class - The ULDataReader is not opened.

Implements

IDataReader.Depth

FieldCount property

Returns the number of columns in the cursor.

Prototypes

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

// C#
public int FieldCount {get;}
```

Return value — The number of columns in the cursor as an integer. Returns 0 if the cursor is closed.

Remarks — This method is identical to the ColumnCount property property.

Implements — IDataRecord.FieldCount

IsBOF property

UL Ext.: Checks whether the current row position is before the first row or not.

Prototypes

```
' Visual Basic
Public Readonly Property IsBOF As Boolean

// C#
public bool IsBOF {get;}
```

Property value — True if the current row position is before the first row, false otherwise.

Exceptions

- ULException class - A SQL error occurred.

IsClosed property

Checks whether the cursor is currently open.

Prototypes

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

// C#
public bool IsClosed {get;}
```

Property value — True if the cursor is currently open, false if the cursor is closed.

Implements — IDataReader.IsClosed

IsEOF property

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

not.

Prototypes

```
' Visual Basic
Public Readonly Property IsEOF As Boolean

// C#
public bool IsEOF {get;}
```

Property value

True if the current row position is after the last row, false otherwise.

Exceptions

- ULException class - A SQL error occurred.

Item property

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

Prototypes

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

// C#
public object Item {get;}
```

Parameters

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

Return value

The column value as the .NET type most appropriate for the column or DBNull if column is NULL.

Remarks

This method is identical in functionality to the GetValue method method.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.Item

See also

- “ULDataReader class” on page 150
- “ULDataReader members” on page 150
- “GetFieldType method” on page 166
- “Item property” on page 155

Item property

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

Prototypes

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

// C#
public object **Item** {get;}

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

Return value
The column value as the .NET type most appropriate for the column or DBNull if column is NULL.

Remarks
Note that in result sets, not all columns have names and not all column names are unique. If you are not using aliases, the name of a non-computed column is prefixed with the name of the table the column is from. For example, MyTable.ID is the name of the only column in the result set for the query "SELECT ID FROM MyTable".

When accessing columns multiple times, it is more efficient to access columns by column ID than by name.

This method is equivalent to:

```
dataReader.GetValue( dataReader.GetOrdinal( name ) )
```

Exceptions
- ULException class - A SQL error occurred.

Implements
IDataRecord.Item

See also
- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "Item property" on page 155
- "GetOrdinal method" on page 171
- "GetValue method" on page 176
- "GetFieldType method" on page 166

RecordsAffected property

Returns the number of rows changed, inserted, or deleted by execution of the SQL statement. For SELECT statements or CommandType.TableDirect tables, this value is -1.

Prototypes

' Visual Basic
NotOverridable Public Readonly Property **RecordsAffected** As Integer _
  Implements IDataReader.RecordsAffected

// C#
public int **RecordsAffected** {get;}

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

Implements
IDataReader.RecordsAffected

RowCount property

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

Prototypes

```
' Visual Basic
Public Readonly Property RowCount As Integer

// C#
public int RowCount {get;}
```

Property value

The number of rows in the cursor.

Remarks

One use for RowCount is to decide when to delete old rows to save space. Old rows can be deleted from the UltraLite database without being deleted from the consolidated database using the StopSynchronizationDelete method method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "StartSynchronizationDelete method" on page 95
- "StopSynchronizationDelete method" on page 96

Schema property

UL Ext.: Holds the schema of this cursor.

Prototypes

```
' Visual Basic
Public Readonly Property Schema As ULCursorSchema

// C#
public ULCursorSchema Schema {get;}
```

Property value

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

Remarks

This property represents the complete schema of the cursor, including UltraLite.NET extended information which is not represented in the results from GetSchemaTable method.

Close method

Closes the cursor.

Prototypes

```
' Visual Basic
NotOverridable Public Sub Close() _
  Implements IDataReader.Close
```

```
// C#
public void Close();
```

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

Exceptions — ♦ ULException class - A SQL error occurred.

Implements — IDataReader.Close

Dispose method

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

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Sub Dispose() _
  Implements IDisposable.Dispose
```

```
// C#
public void Dispose();
```

Exceptions — ♦ ULException class - A SQL error occurred.

Implements — IDisposable.Dispose

Finalize method

Releases the unmanaged resources used by the ULDataReader.

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()
```

```
// C#
protected override void Finalize();
```

GetBoolean method

Returns the value for the specified column as a Boolean.

Prototypes

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

```
// C#
public bool GetBoolean(
  int columnID
);
```

Parameters

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

Return value: The column value as a Boolean.

Exceptions:
- ULException class - A SQL error occurred.

Implements: IDataRecord.GetBoolean

See also:
- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetByte method

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

Prototypes:

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

```
// C#
public byte GetByte(
  int columnID
);
```

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

Return value: The column value as a Byte.

Exceptions:
- ULException class - A SQL error occurred.

Implements: IDataRecord.GetByte

See also:
- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetBytes method

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

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Function GetBytes( _
  ByVal columnID As Integer, _
  ByVal srcOffset As Long, _
  ByVal dst As Byte(), _
  ByVal dstOffset As Integer, _
  ByVal count As Integer _
) As Long _
  Implements IDataRecord.GetBytes

// C#
public long GetBytes(
  int columnID,
  long srcOffset,
  byte[] dst,
  int dstOffset,
  int count
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.
- **srcOffset** The start position in the column value. Zero is the beginning of the value.
- **dst** The destination array.
- **dstOffset** The start position in the destination array.
- **count** The number of bytes to be copied.

Return value

The actual number of bytes copied.

Remarks

If you pass a *dst* buffer that is a null reference (Nothing in Visual Basic), GetBytes returns the length of the field in bytes.

The bytes at position *srcOffset* through *srcOffset*+*count*-1 of the value are copied into positions *dstOffset* through *dstOffset*+*count*-1, respectively, of the destination array. If the end of the value is encountered before *count* bytes are copied, the remainder of the destination array is left unchanged.

If any of the following is true, a ULException class with code ULSQLCode enumeration is thrown and the destination is not modified:

- *srcOffset* is negative.
- *dstOffset* is negative.
- *count* is negative.
- *dstOffset*+*count* is greater than *dst*.Length.

For other errors, a ULException class with the appropriate error code is thrown.

Exceptions
- ULException class - A SQL error occurred.

Implements
IDataRecord.GetBytes

See also
- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166
- "GetBytes method" on page 161

GetBytes method

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

Prototypes

```
' Visual Basic
Overloads Public Function GetBytes( _
  ByVal columnID As Integer _
) As Byte()

// C#
public byte[] GetBytes(
  int columnID
);
```

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

Return value
The column value as an array of Bytes.

Exceptions
- ULException class - A SQL error occurred.

See also
- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166
- "GetBytes method" on page 159

GetChar method

This method is not supported in UltraLite.NET.

Prototypes

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

// C#
public char GetChar(
  int columnID
);
```

Parameters

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

Return value

This method is not supported in UltraLite.NET.

Exceptions

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

Implements

IDataRecord.GetChar

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166
- "GetString method" on page 174

GetChars method

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

Prototypes

```
' Visual Basic
NotOverridable Public Function GetChars( _
  ByVal columnID As Integer, _
  ByVal srcOffset As Long, _
  ByVal dst As Char(), _
  ByVal dstOffset As Integer, _
  ByVal count As Integer _
) As Long _
  Implements IDataRecord.GetChars

// C#
public long GetChars(
  int columnID,
  long srcOffset,
  char[] dst,
  int dstOffset,
  int count
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.
- **srcOffset** The start position in the column value. Zero is the beginning of the value.
- **dst** The destination array.
- **dstOffset** The start position in the destination array.
- **count** The number of characters to be copied.

Return value

The actual number of characters copied.

Remarks

If you pass a *dst* buffer that is a null reference (Nothing in Visual Basic), GetChars returns the length of the field in characters.

The characters at position *srcOffset* through *srcOffset*+*count*-1 of the value are copied into positions *dstOffset* through *dstOffset*+*count*-1, respectively, of the destination array. If the end of the value is encountered before *count* characters are copied, the remainder of the destination array is left unchanged.

If any of the following is true, a ULException class with code ULSQLCode enumeration is thrown and the destination is not modified:

- *srcOffset* is negative.
- *dstOffset* is negative.
- *count* is negative.
- *dstOffset*+*count* is greater than *dst*.Length.

For other errors, a ULException class with the appropriate error code is thrown.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetChars

See also

- “ULDataReader class” on page 150
- “ULDataReader members” on page 150
- “GetOrdinal method” on page 171
- “GetFieldType method” on page 166

GetData method

This method is not supported in UltraLite.NET.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetData( _
  ByVal i As Integer _
) As IDataReader _
  Implements IDataRecord.GetData

// C#
public IDataReader GetData(
  int i
);
```

Parameters

- **i** An integer value.

Return value

This method is not supported in UltraLite.NET.

Exceptions

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

Implements

IDataRecord.GetData

GetDataTypeName method

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

Prototypes

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

// C#
public string  GetDataTypeName(
  int columnID
);
```

Parameters

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

Return value

A string corresponding to the column's ULDbType enumeration.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetDataTypeName

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetColumnULDbType method" on page 117

GetDateTime method

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

Prototypes

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

// C#
public DateTime GetDateTime(
  int columnID
);
```

Parameters

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

Return value

The column value as a DateTime.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetDateTime

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetDecimal method

Returns the value for the specified column as a Decimal.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetDecimal( _
  ByVal columnID As Integer _
) As Decimal _
  Implements IDataRecord.GetDecimal

// C#
public decimal GetDecimal(
  int columnID
);
```

Parameters

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

Return value

The column value as a Decimal.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetDecimal

See also

- "ULDataReader class" on page 150

- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetDouble method

Returns the value for the specified column as a Double.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetDouble( _
  ByVal columnID As Integer _
) As Double _
  Implements IDataRecord.GetDouble

// C#
public double GetDouble(
  int columnID
);
```

Parameters

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

Return value

The column value as a Double.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetDouble

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetFieldType method

Returns the Type most appropriate for the specified column.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetFieldType( _
  ByVal columnID As Integer _
) As Type _
  Implements IDataRecord.GetFieldType

// C#
public Type GetFieldType(
  int columnID
);
```

Parameters

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

Return value: A Type value for the column.

Exceptions:
- ULException class - A SQL error occurred.

Implements: IDataRecord.GetFieldType

See also:
- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetDataTypeName method" on page 164
- "GetColumnULDbType method" on page 117

GetFloat method

Returns the value for the specified column as a Single.

Prototypes:

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

```
// C#
public float  GetFloat(
  int columnID
);
```

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

Return value: The column value as a Single.

Exceptions:
- ULException class - A SQL error occurred.

Implements: IDataRecord.GetFloat

See also:
- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetGuid method

Returns the value for the specified column as a UUID (Guid).

Prototypes

```
' Visual Basic
NotOverridable Public Function GetGuid( _
  ByVal columnID As Integer _
) As Guid _
  Implements IDataRecord.GetGuid

// C#
public Guid GetGuid(
  int columnID
);
```

Parameters

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

Return value

The column value as a Guid.

Remarks

This method is only valid for columns of type ULDbType enumeration or for columns of type ULDbType enumeration with length 16.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetGuid

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166
- "GetColumnULDbType method" on page 117
- "GetColumnSize method" on page 116

GetInt16 method

Returns the value for the specified column as an Int16.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetInt16( _
  ByVal columnID As Integer _
) As Short _
  Implements IDataRecord.GetShort

// C#
public short GetInt16(
  int columnID
);
```

Parameters

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

Return value: The column value as an Int16.

Exceptions:
- ULException class - A SQL error occurred.

Implements: IDataRecord.GetInt16

See also:
- “ULDataReader class” on page 150
- “ULDataReader members” on page 150
- “GetOrdinal method” on page 171
- “GetFieldType method” on page 166

GetInt32 method

Returns the value for the specified column as an Int32.

Prototypes:

```
' Visual Basic
NotOverridable Public Function GetInt32( _
  ByVal columnID As Integer _
) As Integer _
  Implements IDataRecord.GetInteger
```

```
// C#
public int GetInt32(
  int columnID
);
```

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

Return value: The column value as an Int32.

Exceptions:
- ULException class - A SQL error occurred.

Implements: IDataRecord.GetInt32

See also:
- “ULDataReader class” on page 150
- “ULDataReader members” on page 150
- “GetOrdinal method” on page 171
- “GetFieldType method” on page 166

GetInt64 method

Returns the value for the specified column as an Int64.

Prototypes:

```
' Visual Basic
NotOverridable Public Function GetInt64( _
  ByVal columnID As Integer _
) As Long _
  Implements IDataRecord.GetLong
```

```
// C#
public long GetInt64(
  int columnID
);
```

Parameters

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

Return value

The column value as an Int64

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetInt64

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetName method

Returns the name of the specified column.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetName( _
  ByVal columnID As Integer _
) As String _
  Implements IDataRecord.GetName
```

```
// C#
public string  GetName(
  int columnID
);
```

Parameters

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

Return value

The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned.

Remarks

Note that in result sets, not all columns have names and not all column names are unique. If you are not using aliases, the name of a non-computed column is prefixed with the name of the table the column is from. For example, MyTable.ID is the name of the only column in the result set for the query "SELECT ID FROM MyTable".

This method is identical to the GetColumnName method method.

Exceptions

- ULException class - A SQL error occurred.

Implements: IDataRecord.GetName

See also:
- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "FieldCount property" on page 154
- "GetSchemaTable method" on page 172

GetOrdinal method

Returns the column ID of the named column.

Prototypes:

```
' Visual Basic
NotOverridable Public Function GetOrdinal( _
  ByVal columnName As String _
) As Integer _
  Implements IDataRecord.GetOrdinal

// C#
public int GetOrdinal(
  string  columnName
);
```

Parameters:
- **columnName** The name of the column.

Return value: The column ID of the named column.

Remarks: Column IDs range from 0 to FieldCount property-1, inclusively.

Note that in result sets, not all columns have names and not all column names are unique. If you are not using aliases, the name of a non-computed column is prefixed with the name of the table the column is from. For example, MyTable.ID is the name of the only column in the result set for the query "SELECT ID FROM MyTable".

Column IDs and counts may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

This method is identical to the GetColumnID method method.

Exceptions:
- ULException class - A SQL error occurred.

Implements: IDataRecord.GetOrdinal

See also:
- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetSchemaTable method" on page 172

GetSchemaTable method

Returns a DataTable that describes the column metadata of the ULDataReader.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetSchemaTable() As DataTable _
  Implements IDataReader.GetSchemaTable

// C#
public DataTable GetSchemaTable();
```

Return value

A DataTable describing the schema of each column in the ULDataReader.

Remarks

The GetSchemaTable method returns metadata about each column in the following order:

DataTable Column	Description
ColumnName	The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned. Note that in result sets, not all columns have names and not all column names are unique.
ColumnOrdinal	The ID of the column. The value is in the range [0,FieldCount property-1].
ColumnSize	For sized columns, the maximum length of a value in the column. For other columns, this is the size in bytes of the data type.
NumericPrecision	The precision of a numeric column (ProviderType ULDbType enumeration or ULDbType enumeration) or DBNull if the column is not numeric.
NumericScale	The scale of a numeric column (ProviderType ULDbType enumeration or ULDbType enumeration) or DBNull if the column is not numeric.
IsUnique	True if the column is a non-computed unique column in the table (BaseTableName) it is taken from.
IsKey	True if the column is one of a set of columns in the result set that taken together from a unique key for the result set. The set of columns with IsKey set to true does not need to be the minimal set that uniquely identifies a row in the result set.

DataTable Column	Description
BaseCatalogName	The name of the catalog in the database that contains the column. For UltraLite.NET, this value is always DBNull.
BaseColumnName	The original name of the column in the table BaseTableName of the database or DBNull if the column is computed or if this information cannot be determined.
BaseSchemaName	The name of the schema in the database that contains the column. For UltraLite.NET, this value is always DBNull.
BaseTableName	The name of the table in the database that contains the column, or DBNull if column is computed or if this information cannot be determined.
DataType	The .NET data type that is most appropriate for this type of column.
AllowDBNull	True if the column is nullable, false if the column is not nullable or if this information can not be determined.
ProviderType	The ULDbType enumeration of the column.
IsIdentity	True if the column is an identity column, false if it is not an identity column. For UltraLite.NET, this value is always false.
IsAutoIncrement	True if the column is an autoincrement or global autoincrement column, false otherwise (or if this information can not be determined).
IsRowVersion	True if the column contains a persistent row identifier that cannot be written to, and has no meaningful value except to identity the row. For UltraLite.NET, this value is always false.
IsLong	True if the column is a ULDbType enumeration or a ULDbType enumeration column, false otherwise.
IsReadOnly	True if the column is read-only, false if the column is modifiable or if its access cannot be determined.

Implements IDataReader.GetSchemaTable

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "Schema property" on page 157

GetString method

Returns the value for the specified column as a String.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetString( _
  ByVal columnID As Integer _
) As String _
  Implements IDataRecord.GetString

// C#
public string  GetString(
  int columnID
);
```

Parameters

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

Return value

The column value as a String.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetString

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetTimeSpan method

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

Prototypes

```
' Visual Basic
Public Function GetTimeSpan( _
  ByVal columnID As Integer _
) As TimeSpan

// C#
public TimeSpan GetTimeSpan(
  int columnID
);
```

Parameters

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

Return value

The column value as a TimeSpan.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetUInt16 method

Returns the value for the specified column as a UInt16.

Prototypes

```
' Visual Basic
Public Function GetUInt16( _
  ByVal columnID As Integer _
) As UInt16

// C#
public ushort GetUInt16(
  int columnID
);
```

Parameters

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

Return value

The column value as an UInt16.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetUInt32 method

Returns the value for the specified column as a UInt32.

Prototypes

```
' Visual Basic
Public Function GetUInt32( _
  ByVal columnID As Integer _
) As UInt32
```

```
// C#
public uint GetUInt32(
  int columnID
);
```

Parameters

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

Return value

The column value as an UInt32.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULDataReader class” on page 150
- “ULDataReader members” on page 150
- “GetOrdinal method” on page 171
- “GetFieldType method” on page 166

GetUInt64 method

Returns the value for the specified column as a UInt64.

Prototypes

```
' Visual Basic
Public Function GetUInt64( _
  ByVal columnID As Integer _
) As UInt64
```

```
// C#
public ulong GetUInt64(
  int columnID
);
```

Parameters

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

Return value

The column value as a UInt64

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULDataReader class” on page 150
- “ULDataReader members” on page 150
- “GetOrdinal method” on page 171
- “GetFieldType method” on page 166

GetValue method

Returns the value of the specified column in its native format.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetValue( _
  ByVal columnID As Integer _
) As Object _
  Implements IDataRecord.GetValue

// C#
public object GetValue(
  int columnID
);
```

Parameters

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

Return value

The column value as the .NET type most appropriate for the column or DBNull if column is NULL.

Remarks

This method is identical in functionality to the Item property property.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.GetValue

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

GetValues method

Returns all the column values for the current row.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetValues( _
  ByVal values As Object() _
) As Integer _
  Implements IDataRecord.GetValues

// C#
public int GetValues(
  object[] values
);
```

Parameters

- **values** The array of Objects to hold the entire row.

Return value

The number of column values retrieved. If the length of the array is greater than the number of columns (FieldCount property), only FieldCount items are retrieved and the rest of the array is left unchanged.

Remarks

For most applications, the GetValues method provides an efficient means for retrieving all columns, rather than retrieving each column individually.

You can pass an Object array that contains fewer than the number of columns contained in the resulting row. Only the amount of data the Object array holds is copied to the array. You can also pass an Object array whose length is more than the number of columns contained in the resulting row.

This method returns DBNull for NULL database columns. For other columns, it returns the value of the column in its native format.

Exceptions

- ULException class - A SQL error occurred.
- System.ArgumentNullException - The *values* array is NULL or has zero length.

Implements

IDataRecord.GetValues

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "FieldCount property" on page 154
- "GetFieldType method" on page 166
- "GetValue method" on page 176

IsDBNull method

Checks whether the value from the specified column is NULL.

Prototypes

```
' Visual Basic
NotOverridable Public Function IsDBNull( _
  ByVal columnID As Integer _
) As Boolean _
  Implements IDataRecord.IsDBNull

// C#
public bool IsDBNull(
  int columnID
);
```

Parameters

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

Return value

True if value is NULL, false if value is not NULL.

Exceptions

- ULException class - A SQL error occurred.

Implements

IDataRecord.IsDBNull

See also

- "ULDataReader class" on page 150
- "ULDataReader members" on page 150
- "GetOrdinal method" on page 171
- "GetFieldType method" on page 166

MoveAfterLast method

UL Ext.: Positions the cursor to after the last row of the cursor.

Prototypes

' Visual Basic
Public Sub **MoveAfterLast()**

// C#
public void **MoveAfterLast();**

Exceptions

- ULException class - A SQL error occurred.

MoveBeforeFirst method

UL Ext.: Positions the cursor to before the first row of the cursor.

Prototypes

' Visual Basic
Public Sub **MoveBeforeFirst()**

// C#
public void **MoveBeforeFirst();**

Exceptions

- ULException class - A SQL error occurred.

MoveFirst method

UL Ext.: Positions the cursor to the first row of the cursor.

Prototypes

' Visual Basic
Public Function **MoveFirst()** As Boolean

// C#
public bool **MoveFirst();**

Return value

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

Exceptions

- ULException class - A SQL error occurred.

MoveLast method

UL Ext.: Positions the cursor to the last row of the cursor.

Prototypes

' Visual Basic
Public Function **MoveLast()** As Boolean

// C#
public bool **MoveLast();**

Return value

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

Exceptions

- ULException class - A SQL error occurred.

MoveNext method

UL Ext.: Positions the cursor to the next row or after the last row if the cursor was already on the last row.

Prototypes

```
' Visual Basic
Public Function MoveNext() As Boolean

// C#
public bool MoveNext();
```

Return value

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

Remarks

This method is identical to the Read method method.

Exceptions

- ULException class - A SQL error occurred.

MovePrevious method

UL Ext.: Positions the cursor to the previous row or before the first row.

Prototypes

```
' Visual Basic
Public Function MovePrevious() As Boolean

// C#
public bool MovePrevious();
```

Return value

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

Exceptions

- ULException class - A SQL error occurred.

MoveRelative method

UL Ext.: Positions the cursor relative to the current row.

Prototypes

```
' Visual Basic
Public Function MoveRelative( _
  ByVal offset As Integer _
) As Boolean

// C#
public bool MoveRelative(
  int offset
);
```

Parameters

- **offset** The number of rows to move. Negative values correspond to moving backwards.

Return value	True if successful, false otherwise. For example, the method fails if it positions beyond the first or last row.
Remarks	If the row does not exist, the method returns false, and the cursor position is after the last row (IsEOF property) if *offset* is positive, and before the first row (IsBOF property) if the *offset* is negative.
Exceptions	♦ ULException class - A SQL error occurred.

NextResult method

Advances the ULDataReader to the next result when reading the results of batch SQL statements.

Prototypes

```
' Visual Basic
NotOverridable Public Function NextResult() As Boolean _
  Implements IDataReader.NextResult
```

```
// C#
public bool NextResult();
```

Return value	True if there are more result sets, false otherwise. For UltraLite.NET, always returns false.
Remarks	**UL Ext.:** UltraLite.NET does not support batches of SQL statements, hence the ULDataReader is always positioned on the first and only result set. Calling NextResult has no effect.
Exceptions	♦ ULException class - The ULDataReader is not opened.
Implements	IDataReader.NextResult

Read method

Positions the cursor to the next row, or after the last row if the cursor was already on the last row.

Prototypes

```
' Visual Basic
NotOverridable Public Function Read() As Boolean _
  Implements IDataReader.Read
```

```
// C#
public bool Read();
```

Return value	True if successful, false otherwise. For example, the method fails if there are no more rows.
Remarks	This method is identical to the MoveNext method method.
Exceptions	♦ ULException class - A SQL error occurred.
Implements	IDataReader.Read

ULDbType enumeration

Enumerates the UltraLite.NET database data types.

Prototypes

```
' Visual Basic
Public Enum ULDbType
```

```
// C#
public enum ULDbType
```

Remarks

The table below lists which .NET types are compatible with each ULDbType. In the case of integral types, table columns can always be set using smaller integer types, but can also be set using larger types as long as the actual value is within the range of the type.

ULDbType	Compatible .NET type	C# built-in type	Visual Basic built-in type
Binary, **VarBinary**	System.Byte[], or System.Guid if size is 16	byte[]	Byte()
Bit	System.Boolean	bool	Boolean
Char, **VarChar**	System.String	String	String
Date	System.DateTime	DateTime No built-in type.	Date
Double	System.Double	double	Double
LongBinary	System.Byte[]	byte[]	Byte()
LongVarchar	System.String	String	String
Decimal, **Numeric**	System.String	decimal	Decimal
Float, **Real**	System.Single	float	Single
BigInt	System.Int64	long	Long
Integer	System.Int32	int	Integer
SmallInt	System.Int16	short	Short
Time	System.TimeSpan	TimeSpan No built-in type.	TimeSpan No built-in type.
DateTime, **TimeStamp**	System.DateTime	DateTime No built-in type.	Date
TinyInt	System.Byte	byte	Byte
UnsignedBigInt	System.UInt64	ulong	UInt64 No built-in type.

ULDbType	Compatible .NET type	C# built-in type	Visual Basic built-in type
UnsignedInt	System.UInt32	uint	UInt32 No built-in type.
UnsignedSmallInt	System.UInt16	ushort	UInt16 No built-in type.
UniqueIdentifier	System.Guid	Guid No built-in type.	Guid No built-in type.

Binary columns of length 16 are fully compatible with the UniqueIdentifier type.

Members

Member	Description
BigInt	Signed 64-bit integer.
Binary	Binary data, with a specified maximum length. The enumeration values **Binary** and **VarBinary** are aliases of each other.
Bit	1-bit flag.
Char	Character data, with a specified length. In UltraLite.NET, this type always supports Unicode characters. The types **Char** and **VarChar** are fully compatible.
Date	Date information.
DateTime	Timestamp information (date, time). The enumeration values **DateTime** and **TimeStamp** are aliases of each other.
Decimal	Exact numerical data, with a specified precision and scale. The enumeration values **Decimal** and **Numeric** are aliases of each other.
Double	Double precision floating point number (8 bytes).
Float	Single precision floating point number (4 bytes). The enumeration values **Float** and **Real** are aliases of each other.
Integer	Unsigned 32-bit integer.
LongBinary	Binary data, with variable length.
LongVarchar	Character data, with variable length. In UltraLite.NET, this type always supports Unicode characters.
Numeric	Exact numerical data, with a specified precision and scale. The enumeration values **Decimal** and **Numeric** are aliases of each other.
Real	Single precision floating point number (4 bytes). The enumeration values **Float** and **Real** are aliases of each other.

Member	Description
SmallInt	Signed 16-bit integer.
Time	Time information.
TimeStamp	Timestamp information (date, time). The enumeration values **DateTime** and **TimeStamp** are aliases of each other.
TinyInt	Unsigned 8-bit integer.
UniqueIdentifier	Universally Unique Identifier (UUID/GUID).
UnsignedBigInt	Unsigned 64-bit integer.
UnsignedInt	Unsigned 32-bit integer.
UnsignedSmallInt	Unsigned 16-bit integer.
VarBinary	Binary data, with a specified maximum length. The enumeration values **Binary** and **VarBinary** are aliases of each other.
VarChar	Character data, with a specified maximum length. In UltraLite.NET, this type always supports Unicode characters. The types **Char** and **VarChar** are fully compatible.

See also

- “GetFieldType method” on page 166
- “GetDataTypeName method” on page 164
- “GetColumnULDbType method” on page 117

ULException class

Represents a SQL error returned by the UltraLite.NET database.

Prototypes

```
' Visual Basic
Public Class ULException
  Inherits ApplicationException

// C#
public class ULException :
  ApplicationException
```

Remarks

This class is not serializable under the .NET Compact Framework.

ULException members

Public instance constructors

Member	Description
ULException constructor	Creates a ULException with the given error code.

Public instance properties

Member	Description
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the System.Exception instance that caused the current exception.
Message (inherited from Exception)	Gets a message that describes the current exception.
NativeError property	Returns the SQL code returned by the database.
Source property	Returns the name of the provider that generated the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public instance methods

Member	Description
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the System.Exception that is the root cause of one or more subsequent exceptions.
GetObjectData method	Populates a SerializationInfo with the data needed to serialize this ULException.
ToString (inherited from Exception)	Creates and returns a string representation of the current exception.

Protected instance properties

Member	Description
HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

ULException constructor

Creates a ULException with the given error code.

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal code As ULSQLCode, _
  ByVal s1 As String, _
  ByVal s2 As String, _
  ByVal s3 As String _
)

// C#
public ULException(
  ULSQLCode code,
  string  s1,
  string  s2,
  string  s3
);
```

Parameters

- **code** The code of the exception.
- **s1** The first string for the formatted message.
- **s2** The second string for the formatted message.
- **s3** The third string for the formatted message.

Remarks

The message string corresponding to the specified ULSQLCode enumeration is retrieved from the **iAnywhere.Data.UltraLite.resources**

assembly. Resources are searched for, by culture, using the following order: System.Globalization.CultureInfo.CurrentUICulture, then System.Globalization.CultureInfo.CurrentCulture, and finally culture “EN”.

NativeError property

Returns the SQL code returned by the database.

Prototypes

```
' Visual Basic
Public Readonly Property NativeError As ULSQLCode

// C#
public ULSQLCode NativeError {get;}
```

Property value

The ULSQLCode enumeration value returned by the database.

Source property

Returns the name of the provider that generated the error.

Prototypes

```
' Visual Basic
Public Readonly Property Source As String

// C#
public string  Source {get;}
```

Property value

The string value identifying UltraLite.NET as the provider.

GetObjectData method

Populates a SerializationInfo with the data needed to serialize this ULException.

Prototypes

```
' Visual Basic
Overrides Public Sub GetObjectData( _
  ByVal info As System.Runtime.Serialization.SerializationInfo, _
  ByVal context As System.Runtime.Serialization.StreamingContext _
) _
  Implements ISerializable.GetObjectData

// C#
public override void GetObjectData(
  System.Runtime.Serialization.SerializationInfo info,
  System.Runtime.Serialization.StreamingContext context
);
```

Parameters

- **info** The SerializationInfo to populate with data.
- **context** The destination for this serialization.

Remarks	This method is not supported under the .NET Compact Framework.
Implements	ISerializable.GetObjectData

ULIndexSchema class

UL Ext.: Represents the schema of an UltraLite table index.

Prototypes

' Visual Basic
NotInheritable Public Class **ULIndexSchema**

// C#
public sealed class **ULIndexSchema**

Remarks

This class cannot be directly instantiated. Index schemas are created using the PrimaryKey property, GetIndex method, and GetOptimalIndex method methods of the ULTableSchema class class.

ULIndexSchema members

Public instance properties

Member	Description
ColumnCount property	Returns the number of columns in the index.
IsForeignKey property	Checks whether the index is a foreign key.
IsForeignKeyCheckOnCommit property	Checks whether referential integrity for the foreign key is performed on commits or on inserts and updates.
IsForeignKeyNullable property	Checks whether the foreign key is nullable.
IsOpen property	Determines whether the index schema is open or closed.
IsPrimaryKey property	Checks whether the index is the primary key.
IsUniqueIndex property	Checks whether the index is unique.
IsUniqueKey property	Checks whether the index is a unique key.
Name property	Returns the name of the index.
ReferencedIndexName property	The name of the referenced primary index if the index is a foreign key.
ReferencedTableName property	The name of the referenced primary table if the index is a foreign key.

Public instance methods

Member	Description
GetColumnName method	Returns the name of the *colOrdinalInIndex*'th column in this index.
IsColumnDescending method	Checks whether the named column is used in descending order by the index.

Protected instance methods

Member	Description
Finalize method	Releases the unmanaged resources used by ULIndexSchema.

ColumnCount property

Returns the number of columns in the index.

Prototypes

' Visual Basic
Public Readonly Property **ColumnCount** As Short

// C#
public short **ColumnCount** {get;}

Property value

The number of columns in the index.

Remarks

Column ordinals in indexes range from 1 to ColumnCount, 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.

Prototypes

' Visual Basic
Public Readonly Property **IsForeignKey** As Boolean

// C#
public bool **IsForeignKey** {get;}

Property value

True if the index is the foreign key, false if the index is not the foreign key.

Remarks

Columns in a foreign key may reference another table's non-null, unique index.

Exceptions

- ULException class - A SQL error occurred.

IsForeignKeyCheckOnCommit property

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

Prototypes

' Visual Basic
Public Readonly Property **IsForeignKeyCheckOnCommit** As Boolean

// C#
public bool **IsForeignKeyCheckOnCommit** {get;}

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

Exceptions
- ULException class - A SQL error occurred (including index is not a foreign key).

See also
- "ULIndexSchema class" on page 189
- "ULIndexSchema members" on page 189
- "IsForeignKey property" on page 190

IsForeignKeyNullable property

Checks whether the foreign key is nullable.

Prototypes

' Visual Basic
Public Readonly Property **IsForeignKeyNullable** As Boolean

// C#
public bool **IsForeignKeyNullable** {get;}

Property value
True if the foreign key is nullable, false if the foreign key is not nullable.

Exceptions
- ULException class - A SQL error occurred (including index is not a foreign key).

See also
- "ULIndexSchema class" on page 189
- "ULIndexSchema members" on page 189
- "IsForeignKey property" on page 190

IsOpen property

Determines whether the index schema is open or closed.

Prototypes

' Visual Basic
Public Readonly Property **IsOpen** As Boolean

// C#
public bool **IsOpen** {get;}

Property value
True if the index schema is open, otherwise false.

IsPrimaryKey property

Checks whether the index is the primary key.

Prototypes

' Visual Basic
Public Readonly Property **IsPrimaryKey** As Boolean

// C#
public bool **IsPrimaryKey** {get;}

Property value — True if the index is the primary key, false if the index is not the primary key.

Remarks — Columns in the primary key may not be null.

Exceptions
- ULException class - A SQL error occurred.

IsUniqueIndex property

Checks whether the index is unique.

Prototypes

' Visual Basic
Public Readonly Property **IsUniqueIndex** As Boolean

// C#
public bool **IsUniqueIndex** {get;}

Property value — True if the index is unique, false if the index is not unique.

Remarks — Columns in a unique index may be null.

Exceptions
- ULException class - A SQL error occurred.

IsUniqueKey property

Checks whether the index is a unique key.

Prototypes

' Visual Basic
Public Readonly Property **IsUniqueKey** As Boolean

// C#
public bool **IsUniqueKey** {get;}

Property value — True if the index is a unique key, false if the index is not a unique key.

Remarks — Columns in a unique key may not be null.

Exceptions
- ULException class - A SQL error occurred.

Name property

Returns the name of the index.

Prototypes

' Visual Basic
Public Readonly Property **Name** As String

// C#
public string **Name** {get;}

Property value — A string specifying the name of the index.

Exceptions
- ULException class - A SQL error occurred.

ReferencedIndexName property

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

Prototypes

' Visual Basic
Public Readonly Property **ReferencedIndexName** As String

// C#
public string **ReferencedIndexName** {get;}

Property value

A string specifying the name of the referenced primary index.

Exceptions
- ULException class - A SQL error occurred (including index is not a foreign key).

See also
- "ULIndexSchema class" on page 189
- "ULIndexSchema members" on page 189
- "IsForeignKey property" on page 190

ReferencedTableName property

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

Prototypes

' Visual Basic
Public Readonly Property **ReferencedTableName** As String

// C#
public string **ReferencedTableName** {get;}

Property value

A string specifying the name of the referenced primary table.

Exceptions
- ULException class - A SQL error occurred (including index is not a foreign key).

See also
- "ULIndexSchema class" on page 189
- "ULIndexSchema members" on page 189
- "IsForeignKey property" on page 190

Finalize method

Releases the unmanaged resources used by ULIndexSchema.

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

GetColumnName method

Returns the name of the *colOrdinalInIndex*'th column in this index.

Prototypes

```
' Visual Basic
Public Function GetColumnName( _
  ByVal colOrdinalInIndex As Short _
) As String

// C#
public string  GetColumnName(
  short colOrdinalInIndex
);
```

Parameters

- **colOrdinalInIndex** The ordinal of the desired column in the index. The value must be in the range [1,ColumnCount property].

Return value

The name of the column.

Remarks

Column ordinals and count may change during a schema upgrade. Column ordinals from an index are different than the column IDs in a table or another index, even if they refer to the same physical column in a particular table.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULIndexSchema class" on page 189
- "ULIndexSchema members" on page 189
- "ColumnCount property" on page 190

IsColumnDescending method

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

Prototypes

```
' Visual Basic
Public Function IsColumnDescending( _
  ByVal name As String _
) As Boolean

// C#
public bool IsColumnDescending(
  string  name
);
```

Parameters

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

Return value

True if the column is used in descending order, false if the column is used in ascending order.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULIndexSchema class" on page 189

- "ULIndexSchema members" on page 189
- "GetColumnName method" on page 194
- "ColumnCount property" on page 190

ULInfoMessageEventArgs class

Provides data for the InfoMessage event event.

Prototypes

' Visual Basic
NotInheritable Public Class **ULInfoMessageEventArgs**
  Inherits EventArgs

// C#
public sealed class **ULInfoMessageEventArgs** :
  EventArgs

ULInfoMessageEventArgs members

Public instance properties

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

Public instance methods

Member	Description
ToString method	Returns the string representation of the InfoMessage event event.

Message property

The informational or warning message string returned by the database.

Prototypes

' Visual Basic
Public Readonly Property **Message** As String

// C#
public string **Message** {get;}

Property value

A string containing the informational or warning message.

NativeError property

The SQL code corresponding to the informational message or warning

returned by the database.

Prototypes

' Visual Basic
Public Readonly Property **NativeError** As ULSQLCode

// C#
public ULSQLCode **NativeError** {get;}

Property value

An informational or warning ULSQLCode enumeration value.

Source property

The name of the ADO.NET data provider returning the message.

Prototypes

' Visual Basic
Public Readonly Property **Source** As String

// C#
public string **Source** {get;}

Property value

The string "UltraLite.NET Data Provider".

ToString method

Returns the string representation of the InfoMessage event event.

Prototypes

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

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

Property value

The string "UltraLite.NET Data Provider".

Return value

The informational or warning message string.

ULInfoMessageEventHandler delegate

Represents the method that will handle the InfoMessage event event.

Prototypes

```
' Visual Basic
Public Delegate Sub ULInfoMessageEventHandler( _
  ByVal obj As Object, _
  ByVal args As ULInfoMessageEventArgs _
)

// C#
public delegate void ULInfoMessageEventHandler(
  object obj,
  ULInfoMessageEventArgs args
);
```

Parameters

- **obj** The connection sending the event.
- **args** The ULInfoMessageEventArgs class object that contains the event data.

ULParameter class

Represents a parameter to a ULCommand class.

Prototypes

' Visual Basic
NotInheritable Public Class **ULParameter**
  Inherits MarshalByRefObject
  Implements IDbDataParameter, IDataParameter

// C#
public sealed class **ULParameter** :
  MarshalByRefobject, IDbDataParameter, IDataParameter

Remarks

A ULParameter object can be created directly using one of its many constructors, or using the CreateParameter method method. Because of the special treatment of the 0 and 0.0 constants and the way overloaded methods are resolved, it is highly recommended that you explicitly cast constant values to type object when using the ULParameter constructor constructor. For example:

```
' Visual Basic
Dim p As ULParameter = New ULParameter( "", CType( 0, Object ) )

// C#
ULParameter p = new ULParameter( "", (object)0 );
```

Parameters (including those created by CreateParameter method) must be added to a Parameters property collection to be used. All parameters are treated as positional parameters and are used by a command in the order that they were added.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

Implements: IDbDataParameter, IDataParameter

ULParameter members

Public instance constructors

Member	Description
ULParameter constructor	Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.
ULParameter constructor	Initializes a ULParameter object with the specified parameter name and value.

Member	Description
ULParameter constructor	Initializes a ULParameter object with the specified parameter name and data type. This constructor is not recommended; it is provided for compatibility with other data providers.
ULParameter constructor	Initializes a ULParameter object with the specified parameter name and data type. This constructor is not recommended; it is provided for compatibility with other data providers.
ULParameter constructor	Initializes a ULParameter object with the specified parameter name, data type, and length. This constructor is not recommended; it is provided for compatibility with other data providers.
ULParameter constructor	Initializes a ULParameter object with the specified parameter name, data type, length, direction, nullability, numeric precision, numeric scale, source column, source version, and value. This constructor is not recommended; it is provided for compatibility with other data providers.

Public instance properties

Member	Description
DbType property	Specifies the DbType of the parameter
Direction property	A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.
IsNullable property	Specifies whether the parameter accepts null values.
Offset property	Specifies the offset to the Value property property.
ParameterName property	Specifies the name of the parameter.
Precision property	Specifies the maximum number of digits used to represent the Value property property.
Scale property	Specifies the number of decimal places to which Value property is resolved.
Size property	Specifies the maximum size of the data within the column.
SourceColumn property	Specifies the name of the source column mapped to the DataSet and used for loading or returning the value.
SourceVersion property	The DataRowVersion to use when loading Value property.
ULDbType property	Specifies the ULDbType enumeration of the parameter
Value property	Specifies the value of the parameter.

Public instance methods

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

ULParameter constructor

Initializes a ULParameter object with null (Nothing in Visual Basic) as its value.

Prototypes

' Visual Basic
Overloads Public Sub **New()**

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

Example

The following code creates a ULParameter with the value 3 and adds it to a ULCommand class called cmd.

```
' Visual Basic
Dim p As ULParameter = New ULParameter
p.Value = 3
cmd.Parameters.Add( p )

// C#
ULParameter p = new ULParameter();
p.Value = 3;
cmd.Parameters.Add( p );
```

See also

- "ULParameter class" on page 199
- "ULParameter members" on page 199
- "Value property" on page 210
- "ULParameter constructor" on page 201

ULParameter constructor

Initializes a ULParameter object with the specified parameter name and value.

Prototypes

' Visual Basic
Overloads Public Sub **New(** _
  ByVal *parameterName* As String, _
  ByVal *value* As Object _
)

```
// C#
public ULParameter(
  string parameterName,
  object value
);
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string (“”) or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand class.
- **value** An Object that is to be the value of the parameter.

Remarks

Because of the special treatment of the 0 and 0.0 constants and the way overloaded methods are resolved, it is highly recommended that you explicitly cast constant values to type object when using this constructor.

Example

The following code creates a ULParameter with the value 0 and adds it to a ULCommand class called cmd.

```
' Visual Basic
cmd.Parameters.Add( New ULParameter( "", CType( 0, Object ) ) )

// C#
cmd.Parameters.Add( new ULParameter( "", (object)0 ) );
```

See also

- “ULParameter class” on page 199
- “ULParameter members” on page 199
- “ULParameter constructor” on page 201

ULParameter constructor

Initializes a ULParameter object with the specified parameter name and data type. This constructor is not recommended; it is provided for compatibility with other data providers.

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal parameterName As String, _
  ByVal dbType As ULDbType _
)

// C#
public ULParameter(
  string parameterName,
  ULDbType dbType
);
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string (“”) or a null reference (Nothing in Visual Basic) for

this value. In UltraLite.NET, parameter names are not used by ULCommand class.

- **dbType** One of the ULDbType enumeration values.

Remarks

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

See also

- "ULParameter class" on page 199
- "ULParameter members" on page 199
- "ULParameter constructor" on page 201
- "ULParameter constructor" on page 201

ULParameter constructor

Initializes a ULParameter object with the specified parameter name and data type. This constructor is not recommended; it is provided for compatibility with other data providers.

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal parameterName As String, _
  ByVal dbType As ULDbType, _
  ByVal size As Integer _
)

// C#
public ULParameter(
  string  parameterName,
  ULDbType dbType,
  int size
);
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand class.
- **dbType** One of the ULDbType enumeration values.
- **size** The length of the parameter.

Remarks

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

See also

- "ULParameter class" on page 199
- "ULParameter members" on page 199

- "ULParameter constructor" on page 201
- "ULParameter constructor" on page 201

ULParameter constructor

Initializes a ULParameter object with the specified parameter name, data type, and length. This constructor is not recommended; it is provided for compatibility with other data providers.

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal parameterName As String, _
  ByVal dbType As ULDbType, _
  ByVal size As Integer, _
  ByVal sourceColumn As String _
)

// C#
public ULParameter(
  string  parameterName,
  ULDbType dbType,
  int size,
  string  sourceColumn
);
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand class.
- **dbType** One of the ULDbType enumeration values.
- **size** The length of the parameter.
- **sourceColumn** The name of the source column to map.

Remarks

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

See also

- "ULParameter class" on page 199
- "ULParameter members" on page 199
- "ULParameter constructor" on page 201
- "ULParameter constructor" on page 201

ULParameter constructor

Initializes a ULParameter object with the specified parameter name, data type, length, direction, nullability, numeric precision, numeric scale, source

column, source version, and value. This constructor is not recommended; it is provided for compatibility with other data providers.

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal parameterName As String, _
  ByVal dbType As ULDbType, _
  ByVal size As Integer, _
  ByVal direction As ParameterDirection, _
  ByVal isNullable As Boolean, _
  ByVal precision As Byte, _
  ByVal scale As Byte, _
  ByVal sourceColumn As String, _
  ByVal sourceVersion As DataRowVersion, _
  ByVal value As Object _
)
```

```
// C#
public ULParameter(
  string  parameterName,
  ULDbType dbType,
  int size,
  ParameterDirection direction,
  bool isNullable,
  byte precision,
  byte scale,
  string  sourceColumn,
  DataRowVersion sourceVersion,
  object value
);
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand class.
- **dbType** One of the ULDbType enumeration values.
- **size** The length of the parameter.
- **direction** One of the ParameterDirection values.
- **isNullable** True if the value of the field can be null; otherwise, false.
- **precision** The total number of digits to the left and right of the decimal point to which Value is resolved.
- **scale** The total number of decimal places to which Value is resolved.
- **sourceColumn** The name of the source column to map.

- **sourceVersion** One of the DataRowVersion values.
- **value** An Object that is to be the value of the parameter.

Exceptions
- ULException class - Only the ParameterDirection.Input direction is supported in UltraLite.NET.

See also
- “ULParameter class” on page 199
- “ULParameter members” on page 199
- “ULParameter constructor” on page 201
- “ULParameter constructor” on page 201

DbType property

Specifies the DbType of the parameter

Prototypes

```
' Visual Basic
NotOverridable Public Property DbType As DbType _
  Implements IDataParameter.DbType

// C#
public DbType DbType {get;set;}
```

Property value: One of the DbType values.

Remarks: The ULDbType property and DbType properties are linked. Therefore, setting the DbType changes the ULDbType property to a supporting ULDbType enumeration.

Exceptions
- ArgumentException - There is no mapping from the specified value to a ULDbType enumeration, hence, the specified value is not supported.

Implements: IDataParameter.DbType

Direction property

A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

Prototypes

```
' Visual Basic
NotOverridable Public Property Direction As ParameterDirection _
  Implements IDataParameter.Direction

// C#
public ParameterDirection Direction {get;set;}
```

Property value: One of the ParameterDirection values.

Remarks: In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

Exceptions

- ULException class - Only the ParameterDirection.Input direction is supported in UltraLite.NET.

Implements IDataParameter.Direction

IsNullable property

Specifies whether the parameter accepts null values.

Prototypes

```
' Visual Basic
Public Property IsNullable As Boolean _
  Implements IDataParameter.IsNullable

// C#
public bool IsNullable {get;set;}
```

Property value

True if null values are accepted, false otherwise. The default is false. Null values are handled using the DBNull class.

Remarks

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

Implements IDataParameter.IsNullable

Offset property

Specifies the offset to the Value property property.

Prototypes

```
' Visual Basic
Public Property Offset As Integer

// C#
public int Offset {get;set;}
```

Property value

The offset to the value. The default is 0.

Remarks

In UltraLite.NET parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

ParameterName property

Specifies the name of the parameter.

Prototypes

```
' Visual Basic
NotOverridable Public Property ParameterName As String _
  Implements IDataParameter.ParameterName

// C#
public string  ParameterName {get;set;}
```

Property value — A string representing the name of the parameter, or an empty string (“’) for unnamed parameters. Specifying a null reference (Nothing in Visual Basic) results in an empty string being used.

Remarks — In UltraLite.NET, parameter names are not used by ULCommand class. All parameters are treated as positional parameters and are used by a command in the order that they were added.

Implements — IDataParameter.ParameterName

Precision property

Specifies the maximum number of digits used to represent the Value property property.

Prototypes

```
' Visual Basic
NotOverridable Public Property Precision As Byte _
  Implements IDbDataParameter.Precision

// C#
public byte Precision {get;set;}
```

Property value — The maximum number of digits used to represent the Value property property. The default value is 0, which indicates that the data provider sets the precision for the Value property property.

Remarks — In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

Exceptions

- ArgumentException - The value is greater then 38.

Implements — IDbDataParameter.Precision

Scale property

Specifies the number of decimal places to which Value property is resolved.

Prototypes

```
' Visual Basic
NotOverridable Public Property Scale As Byte _
  Implements IDbDataParameter.Scale

// C#
public byte Scale {get;set;}
```

Property value — The number of decimal places to which Value property is resolved. The default is 0.

Remarks — In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

Implements IDbDataParameter.Scale

Size property

Specifies the maximum size of the data within the column.

Prototypes

```
' Visual Basic
NotOverridable Public Property Size As Integer _
  Implements IDbDataParameter.Size

// C#
public int Size {get;set;}
```

Property value The maximum size of the data within the column. The default value is inferred from the parameter value. The Size property is used for binary and string types.

Remarks In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

Implements IDbDataParameter.Size

SourceColumn property

Specifies the name of the source column mapped to the DataSet and used for loading or returning the value.

Prototypes

```
' Visual Basic
NotOverridable Public Property SourceColumn As String _
  Implements IDataParameter.SourceColumn

// C#
public string  SourceColumn {get;set;}
```

Property value A string specifying the name of the source column mapped to the DataSet and used for loading or returning the value.

Remarks In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

Implements IDataParameter.SourceColumn

SourceVersion property

The DataRowVersion to use when loading Value property.

Prototypes

```
' Visual Basic
NotOverridable Public Property SourceVersion As DataRowVersion _
  Implements IDataParameter.SourceVersion
```

```
// C#
public DataRowVersion SourceVersion {get;set;}
```

Implements: IDataParameter.SourceVersion

ULDbType property

Specifies the ULDbType enumeration of the parameter

Prototypes

```
' Visual Basic
Public Property ULDbType As ULDbType
```

```
// C#
public ULDbType ULDbType {get;set;}
```

Property value: One of the ULDbType enumeration values.

Remarks: The ULDbType and DbType property properties are linked. Therefore, setting the ULDbType changes the DbType property to a supporting DbType.

In UltraLite.NET, parameters can only be used as IN parameters and all mapping information is ignored. Only the Value property property is important.

Value property

Specifies the value of the parameter.

Prototypes

```
' Visual Basic
NotOverridable Public Property Value As Object _
  Implements IDataParameter.Value
```

```
// C#
public object Value {get;set;}
```

Property value: An Object that specifies the value of the parameter.

Remarks: The value is sent as-is to the data provider without any type conversion or mapping. When the command is executed, the command attempts to convert the value to the required type, signaling a ULException class with ULSQLCode enumeration if it cannot convert the value.

Implements: IDataParameter.Value

ToString method

Returns the string representation of this instance.

Prototypes

```
' Visual Basic
Overrides Public Function ToString() As String
```

```
// C#
public override string ToString();
```

Return value

The name of the parameter.

ULParameterCollection class

Represents all parameters to a ULCommand class.

Prototypes

```
' Visual Basic
NotInheritable Public Class ULParameterCollection
  Implements IDataParameterCollection, IList, ICollection,
    IEnumerable
```

```
// C#
public sealed class ULParameterCollection :
  IDataParameterCollection, IList,
  ICollection, IEnumerable
```

Remarks

All parameters in the collection are treated as positional parameters and are specified in the same order as the question mark placeholders in the CommandText property. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

There is no constructor for ULParameterCollection. You obtain a ULParameterCollection from the Parameters property property.

Implements: IDataParameterCollection

ULParameterCollection members

Public instance properties

Member	Description
Count property	Returns the number of ULParameter class objects in the collection.
Item property	Returns the ULParameter class at the specified index. In C#, this property is the indexer for the ULParameterCollection class.
Item property	Returns the ULParameter class with the specified name. In C#, this property is the indexer for the ULParameterCollection class.

Public instance methods

Member	Description
Add method	Adds a ULParameter class to the collection.
Add method	Adds a ULParameter class to the collection.

Member	Description
Add method	Adds a new ULParameter class, created using the specified parameter name and value, to the collection.
Add method	Adds a new ULParameter class, created using the specified parameter name and data type, to the collection.
Add method	Adds a new ULParameter class, created using the specified parameter name, data type, and length, to the collection.
Add method	Adds a new ULParameter class, created using the specified parameter name, data type, length, and source column name, to the collection.
Clear method	Removes all the parameters from the collection.
Contains method	Checks whether a ULParameter class exists in the collection.
Contains method	Checks whether a ULParameter class with the specified name exists in the collection.
CopyTo method	Copies ULParameter class objects from the ULParameterCollection to the specified array.
GetEnumerator method	Returns an enumerator for the collection.
IndexOf method	Returns the location of the ULParameter class in the collection.
IndexOf method	Returns the location of the ULParameter class with the specified name in the collection.
Insert method	Inserts an ULParameter class in the collection at the specified index.
Remove method	Removes an ULParameter class from the collection.
RemoveAt method	Removes the parameter at the specified index in the collection.
RemoveAt method	Removes the parameter with the specified name from the collection.

Count property

Returns the number of ULParameter class objects in the collection.

Prototypes

```
' Visual Basic
NotOverridable Public Readonly Property Count As Integer _
  Implements ICollection.Count
```

```
// C#
public int Count {get;}
```

Property value

The number of ULParameter class objects in the collection.

Implements — ICollection.Count

Item property

Returns the ULParameter class at the specified index. In C#, this property is the indexer for the ULParameterCollection class.

Prototypes

```
' Visual Basic
Public Property Item As ULParameter

// C#
public ULParameter Item {get;set;}
```

Parameters

- **index** The zero-based index of the parameter to retrieve. The value must be in the range [0,Count property-1]. The first parameter in the collection has an index value of zero.

Return value — The ULParameter class at the specified index.

Exceptions

- System.IndexOutOfRangeException - The index is invalid.

Item property

Returns the ULParameter class with the specified name. In C#, this property is the indexer for the ULParameterCollection class.

Prototypes

```
' Visual Basic
Public Property Item As ULParameter

// C#
public ULParameter Item {get;set;}
```

Parameters

- **parameterName** The name of the parameter to retrieve.

Return value — The ULParameter class with the specified name.

Exceptions

- System.ArgumentNullException - You cannot set a parameter using a null (Nothing in Visual Basic) parameter name.
- System.IndexOutOfRangeException - There is no parameter with the specified name.

See also

- "ULParameterCollection class" on page 212
- "ULParameterCollection members" on page 212
- "Item property" on page 155
- "GetOrdinal method" on page 171
- "GetValue method" on page 176
- "GetFieldType method" on page 166

Add method

Adds a ULParameter class to the collection.

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Function Add( _
  ByVal value As Object _
) As Integer _
  Implements IList.Add

// C#
public int Add(
  object value
);
```

Parameters

- **value** The ULParameter class object to add to the collection.

Return value

The index of the new ULParameter class object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the CommandText property. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

Exceptions

- ArgumentException - The ULParameter class object can only be added to the collection once.
- System.ArgumentNullException - The value cannot be null (Nothing in Visual Basic).
- System.InvalidCastException - The value specified must be a ULParameter class.

Implements

IList.Add

See also

- “ULParameterCollection class” on page 212
- “ULParameterCollection members” on page 212
- “Add method” on page 215
- “Add method” on page 216

Add method

Adds a ULParameter class to the collection.

Prototypes

```
' Visual Basic
Overloads Public Function Add( _
  ByVal value As ULParameter _
) As ULParameter

// C#
public ULParameter Add(
  ULParameter value
);
```

Parameters

- **value** The ULParameter class object to add to the collection.

Return value

The new ULParameter class object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the CommandText property. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

Exceptions

- ArgumentException - The ULParameter class object can only be added to the collection once.
- System.ArgumentNullException - The value cannot be null (Nothing in Visual Basic).

See also

- "ULParameterCollection class" on page 212
- "ULParameterCollection members" on page 212
- "Add method" on page 216

Add method

Adds a new ULParameter class, created using the specified parameter name and value, to the collection.

Prototypes

```
' Visual Basic
Overloads Public Function Add( _
  ByVal parameterName As String, _
  ByVal value As Object _
) As ULParameter

// C#
public ULParameter Add(
  string  parameterName,
  object value
);
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string (“”) or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand class.
- **value** An Object that is to be the value of the parameter.

Return value

The new ULParameter class object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the CommandText property. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

Because of the special treatment of the 0 and 0.0 constants and the way overloaded methods are resolved, it is highly recommended that you explicitly cast constant values to type object when using this method.

Example

The following code adds a ULParameter with the value 0 to a ULCommand class called cmd.

```
' Visual Basic
cmd.Parameters.Add( "", CType( 0, Object ) )

// C#
cmd.Parameters.Add( "", (object)0 );
```

See also

- “ULParameterCollection class” on page 212
- “ULParameterCollection members” on page 212
- “Add method” on page 215

Add method

Adds a new ULParameter class, created using the specified parameter name and data type, to the collection.

Prototypes

' Visual Basic

```
Overloads Public Function Add( _
  ByVal parameterName As String, _
  ByVal ulDbType As ULDbType _
) As ULParameter
```

```
// C#
public ULParameter Add(
  string  parameterName,
  ULDbType ulDbType
);
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string (“”) or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand class.
- **ulDbType** One of the ULDbType enumeration values.

Return value

The new ULParameter class object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the CommandText property. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- “ULParameterCollection class” on page 212
- “ULParameterCollection members” on page 212
- “Add method” on page 215
- “Add method” on page 216

Add method

Adds a new ULParameter class, created using the specified parameter name, data type, and length, to the collection.

Prototypes

```
' Visual Basic
Overloads Public Function Add( _
  ByVal parameterName As String, _
  ByVal ulDbType As ULDbType, _
  ByVal size As Integer _
) As ULParameter
```

```
// C#
public ULParameter Add(
  string  parameterName,
  ULDbType ulDbType,
  int size
);
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for this value. In UltraLite.NET, parameter names are not used by ULCommand class.
- **ulDbType** One of the ULDbType enumeration values.
- **size** The length of the parameter.

Return value

The new ULParameter class object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the CommandText property. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "ULParameterCollection class" on page 212
- "ULParameterCollection members" on page 212
- "Add method" on page 215
- "Add method" on page 216

Add method

Adds a new ULParameter class, created using the specified parameter name, data type, length, and source column name, to the collection.

Prototypes

```
' Visual Basic
Overloads Public Function Add( _
  ByVal parameterName As String, _
  ByVal ulDbType As ULDbType, _
  ByVal size As Integer, _
  ByVal sourceColumn As String _
) As ULParameter

// C#
public ULParameter Add(
  string  parameterName,
  ULDbType ulDbType,
  int size,
  string  sourceColumn
);
```

Parameters

- **parameterName** The name of the parameter. For unnamed parameters, use an empty string ("") or a null reference (Nothing in Visual Basic) for

this value. In UltraLite.NET, parameter names are not used by ULCommand class.

- **ulDbType** One of the ULDbType enumeration values.
- **size** The length of the parameter.
- **sourceColumn** The name of the source column to map.

Return value

The new ULParameter class object.

Remarks

All parameters in the collection are treated as positional parameters and must be added to the collection in the same order as the corresponding question mark placeholders in the CommandText property. For example, the first parameter in the collection corresponds to the first question mark in the SQL statement, the second parameter in the collection corresponds to the second question mark in the SQL statement, and so on. There must be at least as many question marks in the CommandText property as there are parameters in the collection. Nulls are substituted for missing parameters.

See also

- "ULParameterCollection class" on page 212
- "ULParameterCollection members" on page 212
- "Add method" on page 215
- "Add method" on page 216

Clear method

Removes all the parameters from the collection.

Prototypes

```
' Visual Basic
NotOverridable Public Sub Clear() _
  Implements IList.Clear

// C#
public void Clear();
```

Implements

IList.Clear

Contains method

Checks whether a ULParameter class exists in the collection.

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Function Contains( _
  ByVal value As Object _
) As Boolean _
  Implements IList.Contains
```

```
// C#
public bool Contains(
  object value
);
```

Parameters
- **value** The ULParameter class object to check for.

Return value
True if the collection contains the ULParameter class, false otherwise.

Implements
IList.Contains

See also
- "ULParameterCollection class" on page 212
- "ULParameterCollection members" on page 212
- "Contains method" on page 221

Contains method

Checks whether a ULParameter class with the specified name exists in the collection.

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Function Contains( _
  ByVal value As String _
) As Boolean _
  Implements IDataParameterCollection.Contains
```

```
// C#
public bool Contains(
  string  value
);
```

Parameters
- **value** The name of the parameter to search for.

Return value
True if the collection contains the ULParameter class, false otherwise.

Implements
IDataParameterCollection.Contains

See also
- "ULParameterCollection class" on page 212
- "ULParameterCollection members" on page 212
- "Contains method" on page 220

CopyTo method

Copies ULParameter class objects from the ULParameterCollection to the specified array.

Prototypes

```
' Visual Basic
NotOverridable Public Sub CopyTo( _
  ByVal array As System.Array, _
  ByVal index As Integer _
) _
  Implements ICollection.CopyTo

// C#
public void CopyTo(
  System.Array array,
  int index
);
```

Parameters

- **array** The array into which to copy the ULParameter class objects.
- **index** The starting index of the array.

Implements ICollection.CopyTo

GetEnumerator method

Returns an enumerator for the collection.

Prototypes

```
' Visual Basic
NotOverridable Public Function GetEnumerator() As System.Collections.
        IEnumerator _
  Implements IEnumerable.GetEnumerator

// C#
public System.Collections.IEnumerator GetEnumerator();
```

Return value An ArrayList enumerator enumerating the parameters in the collection.

Implements IEnumerable.GetEnumerator

IndexOf method

Returns the location of the ULParameter class in the collection.

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Function IndexOf( _
  ByVal value As Object _
) As Integer _
  Implements IList.IndexOf

// C#
public int IndexOf(
  object value
);
```

Parameters

- **value** The ULParameter class object to locate.

Return value

The zero-based index of the ULParameter class in the collection or -1 if the parameter is not found.

Exceptions

- System.InvalidCastException - The value specified must be a ULParameter class.

Implements

IList.IndexOf

See also

- "ULParameterCollection class" on page 212
- "ULParameterCollection members" on page 212
- "IndexOf method" on page 223

IndexOf method

Returns the location of the ULParameter class with the specified name in the collection.

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Function IndexOf( _
  ByVal parameterName As String _
) As Integer _
  Implements IDataParameterCollection.IndexOf
```

```
// C#
public int IndexOf(
  string  parameterName
);
```

Parameters

- **parameterName** The name of the parameter to locate.

Return value

The zero-based index of the ULParameter class in the collection or -1 if the parameter is not found.

Implements

IDataParameterCollection.IndexOf

See also

- "ULParameterCollection class" on page 212
- "ULParameterCollection members" on page 212
- "IndexOf method" on page 222

Insert method

Inserts an ULParameter class in the collection at the specified index.

Prototypes

```
' Visual Basic
NotOverridable Public Sub Insert( _
  ByVal index As Integer, _
  ByVal value As Object _
) _
  Implements IList.Insert
```

```
// C#
public void Insert(
   int index,
   object value
);
```

Parameters

- **index** The zero-based index where the parameter is to be inserted within the collection.
- **value** The ULParameter class object to insert.

Exceptions

- System.ArgumentNullException - You cannot set a parameter using a null reference (Nothing in Visual Basic).
- System.IndexOutOfRangeException - The index is invalid.
- System.InvalidCastException - The value specified must be a ULParameter class.

Implements IList.Insert

Remove method

Removes an ULParameter class from the collection.

Prototypes

```
' Visual Basic
NotOverridable Public Sub Remove( _
   ByVal value As Object _
) _
   Implements IList.Remove
```

```
// C#
public void Remove(
   object value
);
```

Parameters

- **value** The ULParameter class object to insert.

Exceptions

- ArgumentException - The collection does not contain the specified parameter.
- System.ArgumentNullException - You cannot set a parameter using a null reference (Nothing in Visual Basic).
- System.InvalidCastException - The value specified must be a ULParameter class.

Implements IList.Remove

RemoveAt method

Removes the parameter at the specified index in the collection.

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Sub RemoveAt( _
  ByVal index As Integer _
) _
  Implements IList.RemoveAt
```

```
// C#
public void RemoveAt(
  int index
);
```

Parameters

- **index** The zero-based index of the parameter to remove. The value must be in the range [0,Count property-1]. The first parameter in the collection has an index value of zero.

Exceptions

- System.IndexOutOfRangeException - The index is invalid.

Implements

IList.RemoveAt

See also

- "ULParameterCollection class" on page 212
- "ULParameterCollection members" on page 212
- "RemoveAt method" on page 225

RemoveAt method

Removes the parameter with the specified name from the collection.

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Sub RemoveAt( _
  ByVal parameterName As String _
) _
  Implements IDataParameterCollection.RemoveAt
```

```
// C#
public void RemoveAt(
  string parameterName
);
```

Parameters

- **parameterName** The name of the parameter to retrieve.

Exceptions

- System.IndexOutOfRangeException - There is no parameter with the specified name.

Implements

IDataParameterCollection.RemoveAt

See also

- "ULParameterCollection class" on page 212
- "ULParameterCollection members" on page 212
- "RemoveAt method" on page 224

ULPublicationSchema class

UL Ext.: Represents the schema of an UltraLite publication.

Prototypes

' Visual Basic
NotInheritable Public Class **ULPublicationSchema**

// C#
public sealed class **ULPublicationSchema**

Remarks

This class cannot be directly instantiated. Publication schemas are created using the GetPublicationSchema method method of the ULDatabaseSchema class class.

UltraLite methods requiring a publication mask actually require a set of publications to check. A set is formed by or'ing the publication masks of individual publications. For example:

```
' Visual Basic
Dim mask As Integer = pub1.Mask Or pub2.Mask

// C#
int mask = pub1.Mask | pub2.Mask;
```

Two special mask values are also provided by this class. SYNC_ALL_DB field corresponds to the entire database. SYNC_ALL_PUBS field corresponds to all publications.

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

See also

- "ULPublicationSchema members" on page 226
- "Mask property" on page 227
- "GetLastDownloadTime method" on page 91
- "CountUploadRows method" on page 85
- "Schema property" on page 81

ULPublicationSchema members

Public static fields (Shared)

Member	Description
SYNC_ALL_DB field	Publication mask corresponding to the entire database.
SYNC_ALL_PUBS field	Publication mask corresponding to all publications.

Public instance properties

Member	Description
IsOpen property	Determines whether the publication schema is open or closed.
Mask property	Returns the publication mask of the publication.
Name property	Returns the name of this publication.

SYNC_ALL_DB field

Publication mask corresponding to the entire database.

Prototypes

' Visual Basic
Public Shared **SYNC_ALL_DB** As Integer

// C#
public const int **SYNC_ALL_DB**;

SYNC_ALL_PUBS field

Publication mask corresponding to all publications.

Prototypes

' Visual Basic
Public Shared **SYNC_ALL_PUBS** As Integer

// C#
public const int **SYNC_ALL_PUBS**;

IsOpen property

Determines whether the publication schema is open or closed.

Prototypes

' Visual Basic
Public Readonly Property **IsOpen** As Boolean

// C#
public bool **IsOpen** {get;}

Property value

True if the publication schema is open, false if the publication schema closed.

Mask property

Returns the publication mask of the publication.

Prototypes

' Visual Basic
Public Readonly Property **Mask** As Integer

// C#
public int **Mask** {get;}

Property value

The publication mask of the publication.

Remarks

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

Exceptions

- ULException class - If a SQL error exception occurs. PUBLICATION_NOT_FOUND is issued if a schema upgrade has deleted or renamed this publication.

Name property

Returns the name of this publication.

Prototypes

' Visual Basic
Public Readonly Property **Name** As String

// C#
public string **Name** {get;}

Property value

A string specifying the name of the publication.

Exceptions

- ULException class - A SQL error occurred. PUBLICATION_NOT_FOUND is issued if a schema upgrade has deleted or renamed the publication.

ULResultSetSchema class

UL Ext.: Represents the schema of an UltraLite result set.

Prototypes

```
' Visual Basic
NotInheritable Public Class ULResultSetSchema
  Inherits ULCursorSchema
```

```
// C#
public sealed class ULResultSetSchema :
  ULCursorSchema
```

Remarks

This class cannot be directly instantiated. A ULResultSetSchema class object is attached to a result set as its Schema property property.

A result set schema is only valid while the data reader is open.

See also

- "ULResultSetSchema members" on page 229
- "ULCommand class" on page 56
- "ULDataReader class" on page 150

ULResultSetSchema members

Public instance properties

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

Public instance methods

Member	Description
GetColumnID method (inherited from ULCursorSchema)	Returns the column ID of the named column.
GetColumnName method (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
GetColumnPrecision method (inherited from ULCursorSchema)	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
GetColumnScale method (inherited from ULCursorSchema)	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).

Member	Description
GetColumnSize method (inherited from ULCursorSchema)	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
GetColumnULDbType method (inherited from ULCursorSchema)	Returns the UltraLite.NET data type of the column identified by the specified column ID.
GetSchemaTable method (inherited from ULCursorSchema)	Returns a DataTable that describes the column schema of the ULDataReader class.

Protected instance methods

Member	Description
Finalize method	Releases the unmanaged resources used by the ULResultSetSchema.

Name property

Returns the name of the cursor.

Prototypes

```
' Visual Basic
Public Readonly Property Name As String

// C#
public string  Name {get;}
```

Property value

The SQL statement that generated the ULResultSetSchema.

Finalize method

Releases the unmanaged resources used by the ULResultSetSchema.

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()

// C#
protected override void Finalize();
```

ULRowUpdatedEventArgs class

Provides data for the RowUpdated event event.

Prototypes

' Visual Basic
NotInheritable Public Class **ULRowUpdatedEventArgs**
  Inherits RowUpdatedEventArgs

// C#
public sealed class **ULRowUpdatedEventArgs** :
  RowUpdatedEventArgs

ULRowUpdatedEventArgs members

Public instance constructors

Member	Description
ULRowUpdatedEventArgs constructor	Initializes a new instance of the ULRowUpdatedEventArgs class.

Public instance properties

Member	Description
Command property	Returns the ULCommand class executed when DbDataAdapter.Update is called.
Errors (inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the RowUpdatedEventArgs.Command was executed.
RecordsAffected property	Returns the number of rows changed, inserted, or deleted by the execution of the SQL statement. For SELECT statements this value is -1.
Row (inherited from RowUpdatedEventArgs)	Gets the DataRow sent through an DbDataAdapter.Update .
StatementType (inherited from RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status (inherited from RowUpdatedEventArgs)	Gets the UpdateStatus of the RowUpdatedEventArgs.Command property.
TableMapping (inherited from RowUpdatedEventArgs)	Gets the DataTableMapping sent through an DbDataAdapter.Update.

ULRowUpdatedEventArgs constructor

Initializes a new instance of the ULRowUpdatedEventArgs class.

Prototypes

```
' Visual Basic
Public Sub New( _
  ByVal row As DataRow, _
  ByVal command As IDbCommand, _
  ByVal statementType As StatementType, _
  ByVal tableMapping As DataTableMapping _
)
```

```
// C#
public ULRowUpdatedEventArgs(
  DataRow row,
  IDbCommand command,
  StatementType statementType,
  DataTableMapping tableMapping
);
```

Parameters

- **row** The DataRow sent through an DbDataAdapter.Update.
- **command** The IDbCommand executed when DbDataAdapter.Update is called.
- **statementType** One of the StatementType values that specifies the type of query executed.
- **tableMapping** The DataTableMapping sent through an DbDataAdapter.Update.

Command property

Returns the ULCommand class executed when DbDataAdapter.Update is called.

Prototypes

```
' Visual Basic
Public Readonly Property Command As ULCommand
```

```
// C#
public ULCommand Command {get;}
```

Property value

The ULCommand class object executed by the update.

Remarks

This is the strongly typed version of RowUpdatedEventArgs.Command.

RecordsAffected property

Returns the number of rows changed, inserted, or deleted by the execution of

the SQL statement. For SELECT statements this value is -1.

Prototypes

' Visual Basic
Public Readonly Property **RecordsAffected** As Integer

// C#
public int **RecordsAffected** {get;}

Property value

The number of rows changed, inserted, or deleted; 0 if no rows were affected or the statement failed; and -1 for SELECT statements.

ULRowUpdatedEventHandler delegate

Represents the method that will handle the RowUpdated event event.

Prototypes

```
' Visual Basic
Public Delegate Sub ULRowUpdatedEventHandler( _
  ByVal sender As Object, _
  ByVal e As ULRowUpdatedEventArgs _
)

// C#
public delegate void ULRowUpdatedEventHandler(
  object sender,
  ULRowUpdatedEventArgs e
);
```

Parameters

- **sender** The connection sending the event.
- **e** The ULRowUpdatedEventArgs class object that contains the event data.

ULRowUpdatingEventArgs class

Provides data for the RowUpdating event event.

Prototypes

```
' Visual Basic
NotInheritable Public Class ULRowUpdatingEventArgs
  Inherits RowUpdatingEventArgs

// C#
public sealed class ULRowUpdatingEventArgs :
  RowUpdatingEventArgs
```

ULRowUpdatingEventArgs members

Public instance constructors

Member	Description
ULRowUpdatingEventArgs constructor	Initializes a new instance of the ULRowUpdatingEventArgs class.

Public instance properties

Member	Description
Command property	Specifies the ULCommand class to execute when performing the DbDataAdapter.Update.
Errors (inherited from RowUpdatingEventArgs)	Gets any errors generated by the .NET Framework data provider when the RowUpdatedEventArgs.Command executes.
Row (inherited from RowUpdatingEventArgs)	Gets the DataRow to send through an DbDataAdapter.Update .
StatementType (inherited from RowUpdatingEventArgs)	Gets the type of SQL statement to execute.
Status (inherited from RowUpdatingEventArgs)	Gets the UpdateStatus of the RowUpdatedEventArgs.Command property.
TableMapping (inherited from RowUpdatingEventArgs)	Gets the DataTableMapping to send through the DbDataAdapter.Update.

ULRowUpdatingEventArgs constructor

Initializes a new instance of the ULRowUpdatingEventArgs class.

Prototypes

```
' Visual Basic
Public Sub New( _
  ByVal row As DataRow, _
  ByVal command As IDbCommand, _
  ByVal statementType As StatementType, _
  ByVal tableMapping As DataTableMapping _
)

// C#
public ULRowUpdatingEventArgs(
  DataRow row,
  IDbCommand command,
  StatementType statementType,
  DataTableMapping tableMapping
);
```

Parameters

- **row** The DataRow to update.
- **command** The IDbCommand to execute during the update.
- **statementType** One of the StatementType values that specifies the type of query executed.
- **tableMapping** The DataTableMapping sent through an DbDataAdapter.Update.

Command property

Specifies the ULCommand class to execute when performing the DbDataAdapter.Update.

Prototypes

```
' Visual Basic
Public Property Command As ULCommand

// C#
public ULCommand Command {get;set;}
```

Property value

The ULCommand class object to execute when updating.

Remarks

This is the strongly typed version of RowUpdatingEventArgs.Command.

ULRowUpdatingEventHandler delegate

Represents the method that will handle the RowUpdating event event.

Prototypes

```
' Visual Basic
Public Delegate Sub ULRowUpdatingEventHandler( _
  ByVal sender As Object, _
  ByVal e As ULRowUpdatingEventArgs _
)

// C#
public delegate void ULRowUpdatingEventHandler(
  object sender,
  ULRowUpdatingEventArgs e
);
```

Parameters

- **sender** The connection sending the event.
- **e** The ULRowUpdatingEventArgs class object that contains the event data.

ULRuntimeType enumeration

UL Ext.: Enumerates the types of UltraLite.NET runtimes.

Prototypes

' Visual Basic
Public Enum **ULRuntimeType**

// C#
public enum **ULRuntimeType**

Members

Member	Description
STANDALONE_UL	Selects the standalone UltraLite.NET runtime. The standalone runtime accesses databases directly. Databases are accessed more quickly this way, but cannot be shared.
UL_ENGINE_CLIENT	Selects the UltraLite engine runtime. The UltraLite.NET engine client communicates with the UltraLite engine to access databases. This means that databases can be shared by different applications.

See also

- "RuntimeType property" on page 138

ULSchemaUpgradeData class

UL Ext.: Returns schema upgrade monitoring data.

Prototypes

' Visual Basic
Public Class **ULSchemaUpgradeData**

// C#
public class **ULSchemaUpgradeData**

See also

- "ULSchemaUpgradeData members" on page 239
- "ULSchemaUpgradeListener interface" on page 241

ULSchemaUpgradeData members

Public instance properties

Member	Description
FinalProgressCount property	Returns an estimate of the final progress count.
ProgressCounter property	Returns the schema upgrade progress counter. This number is an approximation of the progress so far.
State property	Returns the current schema upgrade state.
UpgradeOperations property	Returns the number of schema upgrade operations performed.

FinalProgressCount property

Returns an estimate of the final progress count.

Prototypes

' Visual Basic
Public Readonly Property **FinalProgressCount** As Long

// C#
public long **FinalProgressCount** {get;}

Property value

The estimated final progress count.

See also

- "ULSchemaUpgradeData class" on page 239
- "ULSchemaUpgradeData members" on page 239
- "ULSchemaUpgradeState enumeration" on page 243
- "ProgressCounter property" on page 239

ProgressCounter property

Returns the schema upgrade progress counter. This number is an

approximation of the progress so far.

Prototypes

' Visual Basic
Public Readonly Property **ProgressCounter** As Long

// C#
public long **ProgressCounter** {get;}

Property value

The schema upgrade progress counter as a number between 0 and FinalProgressCount property.

See also

- "ULSchemaUpgradeData class" on page 239
- "ULSchemaUpgradeData members" on page 239
- "ULSchemaUpgradeState enumeration" on page 243

State property

Returns the current schema upgrade state.

Prototypes

' Visual Basic
Public Readonly Property **State** As ULSchemaUpgradeState

// C#
public ULSchemaUpgradeState **State** {get;}

Property value

One of the ULSchemaUpgradeState enumeration values.

UpgradeOperations property

Returns the number of schema upgrade operations performed.

Prototypes

' Visual Basic
Public Readonly Property **UpgradeOperations** As Long

// C#
public long **UpgradeOperations** {get;}

Property value

The number of schema operations performed so far.

Remarks

This number is an approximation of the amount of work done during the schema upgrade. This number starts at zero and increases as the update proceeds. This number is updated more frequently than the progress counter and can be used as a relative measure to compare against other schema upgrades.

See also

- "ULSchemaUpgradeData class" on page 239
- "ULSchemaUpgradeData members" on page 239
- "ULSchemaUpgradeState enumeration" on page 243

ULSchemaUpgradeListener interface

UL Ext.: The listener interface for receiving schema upgrade progress events.

Prototypes

' Visual Basic
Public Interface **ULSchemaUpgradeListener**

// C#
public interface **ULSchemaUpgradeListener**

See also

- "ULSchemaUpgradeListener members" on page 241
- "ApplyFile method" on page 145

ULSchemaUpgradeListener members

Public instance methods

Member	Description
SchemaUpgrading method	Invoked during schema upgrade to inform the user of progress. When the state is ULSchemaUpgradeState enumeration, this method should return true to cancel the schema upgrade or return false to continue. For other states, this method's return value is ignored.

SchemaUpgrading method

Invoked during schema upgrade to inform the user of progress. When the state is ULSchemaUpgradeState enumeration, this method should return true to cancel the schema upgrade or return false to continue. For other states, this method's return value is ignored.

Prototypes

' Visual Basic
Public Function **SchemaUpgrading(** _
  ByVal *data* As ULSchemaUpgradeData _
) As Boolean

// C#
public bool **SchemaUpgrading(**
  ULSchemaUpgradeData *data*
);

Parameters

- **data** A ULSchemaUpgradeData class object containing the latest schema upgrade progress data.

Return value

This method should return false to continue or true to cancel the schema upgrade (the state must be ULSchemaUpgradeState enumeration to cancel).

Remarks	No UltraLite.NET API methods should be invoked during a SchemaUpgrading call.

ULSchemaUpgradeState enumeration

UL Ext.: Enumerates all the states that can occur while upgrading a schema.

Prototypes

' Visual Basic
Public Enum **ULSchemaUpgradeState**

// C#
public enum **ULSchemaUpgradeState**

Remarks

STATE_STARTING is the only state during which the upgrade may be cancelled. If the upgrade is cancelled, you will receive a second event with state **STATE_ERROR**.

Members

Member	Description
STATE_ABORT	The schema upgrade has been aborted and the old database is preserved. This state may occur as the result of a recoverable error or user abort.
STATE_DONE	The schema upgrade completed successfully.
STATE_ERROR	A critical error occurred and the database is unusable.
STATE_LAST	Internally used state not sent to SchemaUpgrading method.
STATE_STARTING	The schema upgrade is starting. This is the only state during which the upgrade may be cancelled. If the upgrade is cancelled, you will receive a second event with state **STATE_ABORT**.
STATE_UPGRADING	The schema upgrade is in progress.

See also

- "ULSchemaUpgradeData class" on page 239

ULServerSyncListener interface

UL Ext.: The listener interface for receiving server synchronization messages.

Prototypes

```
' Visual Basic
Public Interface ULServerSyncListener
```

```
// C#
public interface ULServerSyncListener
```

ULServerSyncListener members

Public instance methods

Member	Description
ServerSyncInvoked method	Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

ServerSyncInvoked method

Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

Prototypes

```
' Visual Basic
Public Sub ServerSyncInvoked( _
  ByVal messageName As String _
)
```

```
// C#
public void ServerSyncInvoked(
  string  messageName
);
```

Parameters

- **messageName** The name of the message sent to the application.

Remarks

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

Example

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

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

Public Class MainWindow
  Inherits System.Windows.Forms.Form
  Implements ULServerSyncListener
  Private conn As ULConnection
  Public Sub New(ByVal args() As String)

    MyBase.New()

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

    'Add any initialization after the InitializeComponent() call
    ULConnection.DatabaseManager.SetServerSyncListener( _
        "myCompany.mymsg", "myCompany.myapp", Me _
      )
    'Create Connection
    ...
  End Sub
  Protected Overrides Sub OnClosing( _
        ByVal e As System.ComponentModel.CancelEventArgs _
      )
    ULConnection.DatabaseManager.SetServerSyncListener( _
        Nothing, Nothing, Nothing _
      )
    MyBase.OnClosing(e)
  End Sub
  Public Sub ServerSyncInvoked(ByVal messageName As String) _
      Implements ULServerSyncListener.ServerSyncInvoked
    Me.Invoke(New EventHandler(AddressOf Me.ServerSyncAction))
  End Sub
  Public Sub ServerSyncAction( _
        ByVal sender As Object, ByVal e As EventArgs _
      )
    ' Do Server sync
    conn.Synchronize()
  End Sub
End Class
// C#
using iAnywhere.Data.UltraLite;
public class Form1 : System.Windows.Forms.Form,
         ULServerSyncListener
{
  private System.Windows.Forms.MainMenu mainMenu1;
  private ULConnection conn;
```

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

    //
    // TODO: Add any constructor code after
    // InitializeComponent call
    //
    ULConnection.DatabaseManager.SetServerSyncListener(
        "myCompnay.mymsg", "myCompany.myapp", this
      );
    // Create connection
    ...
  }
  protected override void Dispose( bool disposing )
  {
    base.Dispose( disposing );
  }
  protected override void OnClosing(
      System.ComponentModel.CancelEventArgs e
    )
  {
    ULConnection.DatabaseManager.SetServerSyncListener(
        null, null, null
      );
    base.OnClosing(e);
  }
  public void ServerSyncInvoked( string messageName )
  {
    this.Invoke( new EventHandler( ServerSyncHandler ) );
  }
  internal void ServerSyncHandler(object sender, EventArgs e)
  {
    conn.Synchronize();
  }
}
```

ULSQLCode enumeration

UL Ext.: Enumerates the SQL codes that may be reported by UltraLite.NET.

Prototypes

' Visual Basic
Public Enum **ULSQLCode**

// C#
public enum **ULSQLCode**

Members

Member	Description
SQLE_AGGREGATES_NOT_-ALLOWED	See "Invalid use of an aggregate function" [*ASA Error Messages*, page 268].
SQLE_ALIAS_NOT_UNIQUE	See "Alias '%1' is not unique" [*ASA Error Messages*, page 122].
SQLE_ALIAS_NOT_YET_-DEFINED	See "Definition for alias '%1' must appear before its first reference" [*ASA Error Messages*, page 206].
SQLE_AMBIGUOUS_INDEX_-NAME	See "Index name '%1' is ambiguous" [*ASA Error Messages*, page 234].
SQLE_ARGUMENT_-CANNOT_BE_NULL	See "Argument %1 of procedure '%2' cannot be null" [*ASA Error Messages*, page 129].
SQLE_BAD_ENCRYPTION_-KEY	See "Incorrect or missing encryption key" [*ASA Error Messages*, page 232].
SQLE_BAD_PARAM_INDEX	See "Input parameter index out of range" [*ASA Error Messages*, page 236].
SQLE_CANNOT_ACCESS_-FILE	See "Cannot access file '%1' – %2" [*ASA Error Messages*, page 135].
SQLE_CANNOT_ACCESS_-SCHEMA_FILE	See "Cannot access schema file '%1"' [*ASA Error Messages*, page 136].
SQLE_CANNOT_CHANGE_-USER_NAME	See "Cannot change synchronization user_name when status of the last upload is unknown." [*ASA Error Messages*, page 138].
SQLE_CANNOT_EXECUTE_-STMT	See "Statement cannot be executed" [*ASA Error Messages*, page 339].
SQLE_CANNOT_MODIFY	See "Cannot modify column '%1' in table '%2"' [*ASA Error Messages*, page 152].
SQLE_CANNOT_REGISTER_-LISTENER	See "The specified listener could not be registered" [*ASA Error Messages*, page 375].

Member	Description
SQLE_COLUMN_-AMBIGUOUS	See “Column ’%1’ found in more than one table – need a correlation name” [*ASA Error Messages*, page 168].
SQLE_COLUMN_CANNOT_-BE_NULL	See “Column ’%1’ in table ’%2’ cannot be NULL” [*ASA Error Messages*, page 169].
SQLE_COLUMN_IN_INDEX	See “Cannot alter a column in an index” [*ASA Error Messages*, page 137].
SQLE_COLUMN_NOT_-FOUND	See “Column ’%1’ not found” [*ASA Error Messages*, page 170].
SQLE_COLUMN_NOT_-FOUND_IN_TABLE	See “Column ’%1’ not found in table ’%2”’ [*ASA Error Messages*, page 170].
SQLE_COMMUNICATIONS_-ERROR	See “Communication error” [*ASA Error Messages*, page 173].
SQLE_CONNECTION_-ALREADY_EXISTS	See “This connection already exists” [*ASA Error Messages*, page 380].
SQLE_CONNECTION_NOT_-FOUND	See “Connection not found” [*ASA Error Messages*, page 176].
SQLE_CONNECTION_-RESTORED	See “UltraLite connection was restored” [*ASA Error Messages*, page 391].
SQLE_CONVERSION_ERROR	See “Cannot convert %1 to a %2” [*ASA Error Messages*, page 139].
SQLE_CORRELATION_-NAME_NOT_FOUND	See “Correlation name ’%1’ not found” [*ASA Error Messages*, page 181].
SQLE_COULD_NOT_FIND_-FUNCTION	See “Could not find ’%1’ in dynamic library ’%2”’ [*ASA Error Messages*, page 184].
SQLE_COULD_NOT_LOAD_-LIBRARY	See “Could not load dynamic library ’%1”’ [*ASA Error Messages*, page 185].
SQLE_CURSOR_ALREADY_-OPEN	See “Cursor already open” [*ASA Error Messages*, page 189].
SQLE_CURSOR_NOT_OPEN	See “Cursor not open” [*ASA Error Messages*, page 191].
SQLE_CURSOR_RESTORED	See “UltraLite cursor (or result set or table) was restored” [*ASA Error Messages*, page 391].
SQLE_CURSOROP_NOT_-ALLOWED	See “Illegal cursor operation attempt” [*ASA Error Messages*, page 228].
SQLE_DATABASE_CREATED	See “The database was created.” [*ASA Error Messages*, page 364].

Member	Description
SQLE_DATABASE_ERROR	See “Internal database error %1 – transaction rolled back” [*ASA Error Messages*, page 241].
SQLE_DATABASE_NAME_-REQUIRED	See “Database name required to start server” [*ASA Error Messages*, page 196].
SQLE_DATABASE_NEW	See “Database created without any schema” [*ASA Error Messages*, page 194].
SQLE_DATABASE_NOT_-CREATED	See “Database creation failed: %1” [*ASA Error Messages*, page 195].
SQLE_DATABASE_-UPGRADE_NOT_POSSIBLE	See “Database upgrade not possible” [*ASA Error Messages*, page 201].
SQLE_DATATYPE_NOT_-ALLOWED	See “Expression has unsupported data type” [*ASA Error Messages*, page 217].
SQLE_DBSPACE_FULL	See “A dbspace has reached its maximum file size” [*ASA Error Messages*, page 117].
SQLE_DEVICE_IO_FAILED	See “File I/O failed for ’%1”’ [*ASA Error Messages*, page 220].
SQLE_DIV_ZERO_ERROR	See “Division by zero” [*ASA Error Messages*, page 208].
SQLE_DOWNLOAD_-CONFLICT	See “Download failed because of conflicts with existing rows.” [*ASA Error Messages*, page 209].
SQLE_DROP_DATABASE_-FAILED	See “An attempt to delete database ’%1’ failed” [*ASA Error Messages*, page 123].
SQLE_DUPLICATE_-FOREIGN_KEY	See “Foreign key ’%1’ for table ’%2’ duplicates an existing foreign key” [*ASA Error Messages*, page 220].
SQLE_DYNAMIC_MEMORY_-EXHAUSTED	See “Dynamic memory exhausted” [*ASA Error Messages*, page 211].
SQLE_ENGINE_ALREADY_-RUNNING	See “Database server already running” [*ASA Error Messages*, page 198].
SQLE_ENGINE_NOT_-MULTIUSER	See “Database server not running in multi-user mode” [*ASA Error Messages*, page 200].
SQLE_ERROR	See “Run time SQL error – %1” [*ASA Error Messages*, page 324].
SQLE_ERROR_CALLING_-FUNCTION	See “Could not allocate resources to call external function” [*ASA Error Messages*, page 182].
SQLE_EXPRESSION_ERROR	See “Invalid expression near ’%1”’ [*ASA Error Messages*, page 249].

Member	Description
SQLE_FOREIGN_KEY_-NAME_NOT_FOUND	See “Foreign key name ’%1’ not found” [*ASA Error Messages*, page 221].
SQLE_IDENTIFIER_TOO_-LONG	See “Identifier ’%1’ too long” [*ASA Error Messages*, page 227].
SQLE_INCORRECT_-VOLUME_ID	See “Incorrect volume ID for ’%1”’ [*ASA Error Messages*, page 233].
SQLE_INDEX_NAME_NOT_-UNIQUE	See “Index name ’%1’ not unique” [*ASA Error Messages*, page 235].
SQLE_INDEX_NOT_FOUND	See “Cannot find index named ’%1”’ [*ASA Error Messages*, page 149].
SQLE_INDEX_NOT_UNIQUE	See “Index ’%1’ for table ’%2’ would not be unique” [*ASA Error Messages*, page 233].
SQLE_INTERRUPTED	See “Statement interrupted by user” [*ASA Error Messages*, page 341].
SQLE_INVALID_DATABASE	See “Specified database is invalid” [*ASA Error Messages*, page 335].
SQLE_INVALID_FOREIGN_-KEY	See “No primary key value for foreign key ’%1’ in table ’%2”’ [*ASA Error Messages*, page 287].
SQLE_INVALID_FOREIGN_-KEY_DEF	See “Column ’%1’ in foreign key has a different definition than primary key” [*ASA Error Messages*, page 169].
SQLE_INVALID_GROUP_-SELECT	See “Function or column reference to ’%1’ must also appear in a GROUP BY” [*ASA Error Messages*, page 223].
SQLE_INVALID_OPTION_-SETTING	See “Invalid setting for option ’%1”’ [*ASA Error Messages*, page 263].
SQLE_INVALID_ORDER	See “Invalid ORDER BY specification” [*ASA Error Messages*, page 255].
SQLE_INVALID_PARAMETER	See “Invalid parameter” [*ASA Error Messages*, page 256].
SQLE_INVALID_PARSE_-PARAMETER	See “Parse error: %1” [*ASA Error Messages*, page 300].
SQLE_INVALID_PASSWORD	See “Invalid user ID or password” [*ASA Error Messages*, page 269].
SQLE_INVALID_SQL_-IDENTIFIER	See “Invalid SQL identifier” [*ASA Error Messages*, page 264].
SQLE_LOCKED	See “User ’%1’ has the row in ’%2’ locked” [*ASA Error Messages*, page 413].

Member	Description
SQLE_MEMORY_ERROR	See “Memory error – transaction rolled back” [*ASA Error Messages*, page 279].
SQLE_METHOD_CANNOT_-BE_CALLED	See “Method ’%1’ cannot be called at this time” [*ASA Error Messages*, page 279].
SQLE_NAME_NOT_UNIQUE	See “Item ’%1’ already exists” [*ASA Error Messages*, page 271].
SQLE_NO_COLUMN_NAME	See “Derived table ’%1’ has no name for column %2” [*ASA Error Messages*, page 206].
SQLE_NO_CURRENT_ROW	See “No current row of cursor” [*ASA Error Messages*, page 286].
SQLE_NO_INDICATOR	See “No indicator variable provided for NULL result” [*ASA Error Messages*, page 287].
SQLE_NO_MATCHING_-SELECT_ITEM	See “The Select list for the derived table ’%1’ has no expression to match ’%2”’ [*ASA Error Messages*, page 372].
SQLE_NO_PRIMARY_KEY	See “Table ’%1’ has no primary key” [*ASA Error Messages*, page 351].
SQLE_NOERROR	SQLE_NOERROR(0) - This code indicates that there was no error or warning.
SQLE_NOT_IMPLEMENTED	See “Feature ’%1’ not implemented” [*ASA Error Messages*, page 218].
SQLE_NOT_SUPPORTED_-IN_ULTRALITE	See “Feature not available with UltraLite” [*ASA Error Messages*, page 219].
SQLE_NOTFOUND	See “Row not found” [*ASA Error Messages*, page 323].
SQLE_OVERFLOW_ERROR	See “Value %1 out of range for destination” [*ASA Error Messages*, page 421].
SQLE_PAGE_SIZE_INVALID	See “Invalid database page size” [*ASA Error Messages*, page 246].
SQLE_PERMISSION_DENIED	See “Permission denied: %1” [*ASA Error Messages*, page 303].
SQLE_PRIMARY_KEY_NOT_-UNIQUE	See “Primary key for table ’%1’ is not unique” [*ASA Error Messages*, page 306].
SQLE_PRIMARY_KEY_-TWICE	See “Table cannot have two primary keys” [*ASA Error Messages*, page 355].
SQLE_PRIMARY_KEY_-VALUE_REF	See “Primary key for row in table ’%1’ is referenced by foreign key ’%2’ in table ’%3”’ [*ASA Error Messages*, page 305].
SQLE_PUBLICATION_NOT_-FOUND	See “Publication ’%1’ not found” [*ASA Error Messages*, page 309].

Member	Description
SQLE_RESOURCE_- GOVERNOR_EXCEEDED	See “Resource governor for ’%1’ exceeded” [*ASA Error Messages*, page 316].
SQLE_ROW_DELETED_TO_- MAINTAIN_REFERENTIAL_- INTEGRITY	See “Row was dropped from table %1 to maintain referential integrity” [*ASA Error Messages*, page 324].
SQLE_ROW_DROPPED_- DURING_SCHEMA_- UPGRADE	See “A row could not be converted to the new schema format” [*ASA Error Messages*, page 118].
SQLE_SCHEMA_UPGRADE_- NOT_ALLOWED	See “A schema upgrade is not currently allowed” [*ASA Error Messages*, page 119].
SQLE_SERVER_- SYNCHRONIZATION_ERROR	See “Synchronization failed due to an error on the server: %1” [*ASA Error Messages*, page 345].
SQLE_START_STOP_- DATABASE_DENIED	See “Request to start/stop database denied” [*ASA Error Messages*, page 315].
SQLE_STRING_RIGHT_- TRUNCATION	See “Right truncation of string data” [*ASA Error Messages*, page 320].
SQLE_SUBQUERY_SELECT_- LIST	See “Subquery allowed only one select list item” [*ASA Error Messages*, page 343].
SQLE_SYNC_INFO_INVALID	See “Information for synchronization is incomplete or invalid, check ’%1”’ [*ASA Error Messages*, page 236].
SQLE_SYNC_STATUS_- UNKNOWN	See “The status of the last synchronization upload is unknown” [*ASA Error Messages*, page 376].
SQLE_SYNCHRONIZATION_- NOT_FOUND	See “Cannot find synchronization subscription with the name ’%1”’ [*ASA Error Messages*, page 150].
SQLE_SYNTAX_ERROR	See “Syntax error near ’%1’ %2” [*ASA Error Messages*, page 348].
SQLE_TABLE_HAS_- PUBLICATIONS	See “Table ’%1’ has publications” [*ASA Error Messages*, page 352].
SQLE_TABLE_IN_USE	See “Table in use” [*ASA Error Messages*, page 356].
SQLE_TABLE_NOT_FOUND	See “Table ’%1’ not found” [*ASA Error Messages*, page 354].
SQLE_TOO_MANY_- CONNECTIONS	See “Database server connection limit exceeded” [*ASA Error Messages*, page 199].
SQLE_TOO_MANY_TEMP_- TABLES	See “too many temporary tables in connection” [*ASA Error Messages*, page 386].

Member	Description
SQLE_ULTRALITE_-DATABASE_NOT_FOUND	See “The database ’%1’ was not found” [*ASA Error Messages*, page 363].
SQLE_ULTRALITE_OBJ_-CLOSED	See “Invalid operation on a closed ’%1”’ [*ASA Error Messages*, page 254].
SQLE_ULTRALITE_-RUNTIME_LIBRARY_-MISMATCH	See “UltraLite runtime library is incompatible with the database file” [*ASA Error Messages*, page 392].
SQLE_UNABLE_TO_-CONNECT	See “Database cannot be started – %1” [*ASA Error Messages*, page 194].
SQLE_UNABLE_TO_START_-DATABASE	See “Unable to start specified database: %1” [*ASA Error Messages*, page 396].
SQLE_UNABLE_TO_START_-DATABASE_VER_NEWER	See “Unable to start specified database: Server must be upgraded to start database %1” [*ASA Error Messages*, page 404].
SQLE_UNCOMMITTED_-TRANSACTIONS	See “You cannot synchronize or upgrade with uncommitted transactions” [*ASA Error Messages*, page 430].
SQLE_UNKNOWN_FUNC	See “Unknown function ’%1”’ [*ASA Error Messages*, page 408].
SQLE_UNKNOWN_USERID	See “User ID ’%1’ does not exist” [*ASA Error Messages*, page 416].
SQLE_UNRECOGNIZED_-OPTION	See “The option ’%1’ is not recognized.” [*ASA Error Messages*, page 370].
SQLE_UNSUPPORTED_-CHARACTER_SET_ERROR	See “Database server cannot convert data from/to character set ’%1”’ [*ASA Error Messages*, page 198].
SQLE_UPLOAD_FAILED_-AT_SERVER	See “Synchronization server failed to commit the upload” [*ASA Error Messages*, page 348].
SQLE_WRONG_NUM_OF_-INSERT_COLS	See “Wrong number of values for INSERT” [*ASA Error Messages*, page 427].
SQLE_WRONG_-PARAMETER_COUNT	See “Wrong number of parameters to function ’%1”’ [*ASA Error Messages*, page 427].

ULStreamErrorCode enumeration

UL Ext.: Enumerates the error codes that may be reported by streams during synchronization.

Prototypes

' Visual Basic
Public Enum **ULStreamErrorCode**

// C#
public enum **ULStreamErrorCode**

Members

Member	Description
ACTSYNC_NO_PORT	ACTSYNC_NO_PORT(75)
ACTSYNC_NOT_INSTALLED	ACTSYNC_NOT_INSTALLED(76)
CREATE_RANDOM_OBJECT	CREATE_RANDOM_OBJECT(17)
DEQUEUING_CONNECTION	DEQUEUING_CONNECTION(19)
END_READ	END_READ(11)
END_WRITE	END_WRITE(10)
GENERATE_RANDOM	GENERATE_RANDOM(14)
HTTP_BAD_STATUS_CODE	HTTP_BAD_STATUS_CODE(86)
HTTP_BUFFER_SIZE_OUT_-OF_RANGE	HTTP_BUFFER_SIZE_OUT_OF_RANGE(79)
HTTP_CHUNK_LEN_BAD_-CHARACTER	HTTP_CHUNK_LEN_BAD_CHARACTER(85)
HTTP_CHUNK_LEN_-ENCODED_MISSING	HTTP_CHUNK_LEN_ENCODED_MISSING(84)
HTTP_CLIENT_ID_NOT_SET	HTTP_CLIENT_ID_NOT_SET(78)
HTTP_CONTENT_TYPE_-NOT_SPECIFIED	HTTP_CONTENT_TYPE_NOT_SPECIFIED(77)
HTTP_CRLF_ENCODED_-MISSING	HTTP_CRLF_ENCODED_MISSING(81)
HTTP_CRLF_MISSING	HTTP_CRLF_MISSING(82)
HTTP_EXPECTED_POST	HTTP_EXPECTED_POST(89)
HTTP_EXTRA_DATA_END_-READ	HTTP_EXTRA_DATA_END_READ(80)

Member	Description
HTTP_NO_CONTD_-CONNECTION	HTTP_NO_CONTD_CONNECTION(83)
HTTP_UNABLE_TO_PARSE_-COOKIE	HTTP_UNABLE_TO_PARSE_COOKIE(88)
HTTP_UNKNOWN_-TRANSFER_ENCODING	HTTP_UNKNOWN_TRANSFER_ENCODING(87)
HTTP_VERSION	HTTP_VERSION(54)
INIT_RANDOM	INIT_RANDOM(15)
LOAD_NETWORK_LIBRARY	LOAD_NETWORK_LIBRARY(74)
MEMORY_ALLOCATION	MEMORY_ALLOCATION(6)
NONE	NONE(0)
NOT_IMPLEMENTED	NOT_IMPLEMENTED(12)
PARAMETER	PARAMETER(1)
PARAMETER_NOT_-BOOLEAN	PARAMETER_NOT_BOOLEAN(4)
PARAMETER_NOT_HEX	PARAMETER_NOT_HEX(5)
PARAMETER_NOT_UINT32	PARAMETER_NOT_UINT32(2)
PARAMETER_NOT_UINT32_-RANGE	PARAMETER_NOT_UINT32_RANGE(3)
PARSE	PARSE(7)
READ	READ(8)
SECURE_ADD_CERTIFICATE	SECURE_ADD_CERTIFICATE(39)
SECURE_ADD_TRUSTED_-CERTIFICATE	SECURE_ADD_TRUSTED_CERTIFICATE(48)
SECURE_CERTIFICATE_-CHAIN_FUNC	SECURE_CERTIFICATE_CHAIN_FUNC(28)
SECURE_CERTIFICATE_-CHAIN_LENGTH	SECURE_CERTIFICATE_CHAIN_LENGTH(22)
SECURE_CERTIFICATE_-CHAIN_REF	SECURE_CERTIFICATE_CHAIN_REF(29)

Member	Description
SECURE_CERTIFICATE_-COMMON_NAME	SECURE_CERTIFICATE_COMMON_NAME(52)
SECURE_CERTIFICATE_-COMPANY_NAME	SECURE_CERTIFICATE_COMPANY_NAME(21)
SECURE_CERTIFICATE_-COMPANY_UNIT	SECURE_CERTIFICATE_COMPANY_UNIT(51)
SECURE_CERTIFICATE_-COUNT	SECURE_CERTIFICATE_COUNT(42)
SECURE_CERTIFICATE_-EXPIRED	SECURE_CERTIFICATE_EXPIRED(50)
SECURE_CERTIFICATE_-EXPIRY_DATE	SECURE_CERTIFICATE_EXPIRY_DATE(37)
SECURE_CERTIFICATE_-FILE_NOT_FOUND	SECURE_CERTIFICATE_FILE_NOT_FOUND(33)
SECURE_CERTIFICATE_-NOT_TRUSTED	SECURE_CERTIFICATE_NOT_TRUSTED(24)
SECURE_CERTIFICATE_REF	SECURE_CERTIFICATE_REF(23)
SECURE_CERTIFICATE_-ROOT	SECURE_CERTIFICATE_ROOT(20)
SECURE_CREATE_-CERTIFICATE	SECURE_CREATE_CERTIFICATE(43)
SECURE_CREATE_PRIVATE_-KEY_OBJECT	SECURE_CREATE_PRIVATE_KEY_OBJECT(49)
SECURE_DUPLICATE_-CONTEXT	SECURE_DUPLICATE_CONTEXT(25)
SECURE_ENABLE_NON_-BLOCKING	SECURE_ENABLE_NON_BLOCKING(30)
SECURE_EXPORT_-CERTIFICATE	SECURE_EXPORT_CERTIFICATE(38)
SECURE_HANDSHAKE	SECURE_HANDSHAKE(53)
SECURE_IMPORT_-CERTIFICATE	SECURE_IMPORT_CERTIFICATE(44)

Member	Description
SECURE_READ_-CERTIFICATE	SECURE_READ_CERTIFICATE(34)
SECURE_READ_PRIVATE_-KEY	SECURE_READ_PRIVATE_KEY(35)
SECURE_SET_CHAIN_-NUMBER	SECURE_SET_CHAIN_NUMBER(32)
SECURE_SET_CIPHER_-SUITES	SECURE_SET_CIPHER_SUITES(31)
SECURE_SET_IO	SECURE_SET_IO(26)
SECURE_SET_IO_-SEMANTICS	SECURE_SET_IO_SEMANTICS(27)
SECURE_SET_PRIVATE_KEY	SECURE_SET_PRIVATE_KEY(36)
SECURE_SET_PROTOCOL_-SIDE	SECURE_SET_PROTOCOL_SIDE(47)
SECURE_SET_RANDOM_-FUNC	SECURE_SET_RANDOM_FUNC(46)
SECURE_SET_RANDOM_REF	SECURE_SET_RANDOM_REF(45)
SECURE_SET_READ_FUNC	SECURE_SET_READ_FUNC(55)
SECURE_SET_WRITE_FUNC	SECURE_SET_WRITE_FUNC(56)
SECURE_TRUSTED_-CERTIFICATE_FILE_NOT_-FOUND	SECURE_TRUSTED_CERTIFICATE_FILE_NOT_FOUND(40)
SECURE_TRUSTED_-CERTIFICATE_READ	SECURE_TRUSTED_CERTIFICATE_READ(41)
SEED_RANDOM	SEED_RANDOM(16)
SHUTTING_DOWN	SHUTTING_DOWN(18)
SOCKET_BIND	SOCKET_BIND(62)
SOCKET_CLEANUP	SOCKET_CLEANUP(63)
SOCKET_CLOSE	SOCKET_CLOSE(64)
SOCKET_CONNECT	SOCKET_CONNECT(65)
SOCKET_CREATE_TCPIP	SOCKET_CREATE_TCPIP(60)

Member	Description
SOCKET_CREATE_UDP	SOCKET_CREATE_UDP(61)
SOCKET_GET_HOST_BY_-ADDR	SOCKET_GET_HOST_BY_ADDR(58)
SOCKET_GET_NAME	SOCKET_GET_NAME(66)
SOCKET_GET_OPTION	SOCKET_GET_OPTION(67)
SOCKET_HOST_NAME_-NOT_FOUND	SOCKET_HOST_NAME_NOT_FOUND(57)
SOCKET_LISTEN	SOCKET_LISTEN(69)
SOCKET_LOCALHOST_-NAME_NOT_FOUND	SOCKET_LOCALHOST_NAME_NOT_FOUND(59)
SOCKET_PORT_OUT_OF_-RANGE	SOCKET_PORT_OUT_OF_RANGE(73)
SOCKET_SELECT	SOCKET_SELECT(71)
SOCKET_SET_OPTION	SOCKET_SET_OPTION(68)
SOCKET_SHUTDOWN	SOCKET_SHUTDOWN(70)
SOCKET_STARTUP	SOCKET_STARTUP(72)
WOULD_BLOCK	WOULD_BLOCK(13)
WRITE	WRITE(9)

See also ♦ “StreamErrorCode property” on page 286

ULStreamErrorContext enumeration

UL Ext.: Enumerates the basic network operation being performed when the stream errors occured.

Prototypes

' Visual Basic
Public Enum **ULStreamErrorContext**

// C#
public enum **ULStreamErrorContext**

Members

Member	Description
CLOSE	CLOSE(6)
CREATE	CREATE(3)
DESTROY	DESTROY(4)
END_READ	END_READ(11)
END_WRITE	END_WRITE(10)
GETVALUE	GETVALUE(14)
OPEN	OPEN(5)
READ	READ(7)
REGISTER	REGISTER(1)
SOFTSHUTDOWN	SOFTSHUTDOWN(13)
UNKNOWN	UNKNOWN(0)
UNREGISTER	UNREGISTER(2)
WRITE	WRITE(8)
WRITE_FLUSH	WRITE_FLUSH(9)
YIELD	YIELD(12)

See also

- "StreamErrorContext property" on page 286

ULStreamErrorID enumeration

UL Ext.: Enumerates the network layers that may report errors during synchronization.

Prototypes

' Visual Basic
Public Enum **ULStreamErrorID**

// C#
public enum **ULStreamErrorID**

Members

Member	Description
ACTIVESYNC	ACTIVESYNC(22)
CERTICOM	CERTICOM(11)
CERTICOM_SSL	CERTICOM_SSL(13)
CERTICOM_TLS	CERTICOM_TLS(14)
DH_CAST	DH_CAST(9)
EMAIL	EMAIL(20)
FAKE	FAKE(2)
FILE	FILE(21)
HTTP	HTTP(7)
HTTPS	HTTPS(8)
JAVA_CERTICOM	JAVA_CERTICOM(12)
JAVA_RSA	JAVA_RSA(24)
NETTECH	NETTECH(5)
PALM_CONDUIT	PALM_CONDUIT(3)
PALM_SS	PALM_SS(4)
REPLAY	REPLAY(17)
RIMBB	RIMBB(6)
RSA_TLS	RSA_TLS(23)
SECURE	SECURE(10)
SERIAL	SERIAL(1)
STRM	STRM(18)

Member	Description
TCPIP	TCPIP(0)
UDP	UDP(19)
WIRELESS	WIRELESS(16)
WIRESTRM	WIRESTRM(15)

See also

- "StreamErrorID property" on page 287

ULStreamType enumeration

UL Ext.: Enumerates the types of MobiLink synchronization streams to use for synchronization.

Prototypes

' Visual Basic
Public Enum **ULStreamType**

// C#
public enum **ULStreamType**

Remarks

For information on configuring specific stream types, refer to "Network protocol options for UltraLite synchronization clients" [*MobiLink Clients*, page 341].

Members

Member	Description
ACTIVE_SYNC	ActiveSync synchronization (Windows CE only). UltraLite applications should only use ActiveSync synchronization when notified to do so by the MobiLink provider for ActiveSync. An application can listen for such notification by implementing a ULActiveSyncListener interface and calling SetActiveSyncListener method.
HTTP	Synchronize via HTTP. The HTTP stream uses TCP/IP as its underlying transport. UltraLite applications act as Web browsers and the MobiLink synchronization server acts as a Web server. UltraLite applications send POST requests to send data to the server and GET requests to read data from the server.
HTTPS	Synchronize via HTTPS (HTTP with RSA transport-layer security). Transport-layer security is a separately-licensable component and must be ordered before you can install it. To order this component, see the card in your SQL Anywhere Studio package, or see *http://www.sybase.com/detail?id=1015780*.
TCPIP	Synchronize via TCP/IP.
UNKNOWN	Unknown or no synchronization stream. The user has not set the stream type.

See also

- "Stream property" on page 270

ULSyncParms class

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

Prototypes

' Visual Basic
NotInheritable Public Class **ULSyncParms**

// C#
public sealed class **ULSyncParms**

Remarks

This class cannot be directly instantiated. Each connection has its own ULSyncParms instance, attached as its SyncParms property property.

At most, only one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) may be specified at a time. If more than one of these parameters is set to true, a ULSQLCode enumeration SQLException is thrown by Synchronize method.

Other sources of ULSQLCode enumeration errors include not specifying a Stream property value or a Version property value.

See also

- "ULSyncParms members" on page 263
- "ULConnection class" on page 72
- "SyncParms property" on page 81
- "Synchronize method" on page 96

ULSyncParms members

Public instance properties

Member	Description
AuthenticationParms property	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
CheckpointStore property	Specifies whether the client should perform extra store checkpoints to control the growth of the database store during synchronization.
DisableConcurrency property	Specifies whether concurrent access to UltraLite while performing a synchronization.
DownloadOnly property	Specifies whether to disable or enable uploads when synchronizing.
KeepPartialDownload property	Specifies whether to disable or enable partial downloads when synchronizing.
NewPassword property	Specifies a new MobiLink password for the user specified with UserName.

Member	Description
Password property	The MobiLink password for the user specified by UserName.
PingOnly property	Specifies whether the client should only ping the MobiLink synchronization server instead of performing a real synchronization.
PublicationMask property	Specifies the publications to be synchronized.
ResumePartialDownload property	Specifies whether to resume or discard a previous partial download.
SendColumnNames property	Specifies whether the client should send column names to the MobiLink synchronization server during synchronization.
SendDownloadAck property	Specifies whether the client should send a download acknowledgement to the MobiLink synchronization server during synchronization.
Stream property	Specifies the MobiLink synchronization stream to use for synchronization.
StreamParms property	Specifies the parameters to configure the synchronization stream.
UploadOnly property	Specifies whether to disable or enable downloads when synchronizing.
UserName property	The user name that uniquely identifies the MobiLink client to the MobiLink synchronization server.
Version property	Specifies which synchronization script to use.

Public instance methods

Member	Description
CopyFrom method	Copies the properties of the specified ULSyncParms class object to this ULSyncParms class object.

Protected instance methods

Member	Description
Finalize method	Releases unmanaged resources and performs other cleanup operations before the ULSyncParms is reclaimed by garbage collection.

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Prototypes

' Visual Basic
Public Property **AuthenticationParms** As String()

// C#
public string [] **AuthenticationParms** {get;set;}

Property value

An array of strings, each containing an authentication parameter (null array entries result in a synchronization error). The default is a null reference (Nothing in Visual Basic), meaning no authentication parameters.

Remarks

Only the first 255 strings are used and each string should be no longer than 128 characters (longer strings are truncated when sent to the MobiLink synchronization server).

CheckpointStore property

Specifies whether the client should perform extra store checkpoints to control the growth of the database store during synchronization.

Prototypes

' Visual Basic
Public Property **CheckpointStore** As Boolean

// C#
public bool **CheckpointStore** {get;set;}

Property value

True to specify that the client should perform extra store checkpoints. The default is false, meaning only required checkpointing is done.

Remarks

The checkpoint operation adds I/O operations for the application, and so slows synchronization. This option is most useful for large downloads with many updates. Devices with slow flash memory may not want to pay the performance penalty associated with additional checkpoints.

DisableConcurrency property

Specifies whether concurrent access to UltraLite while performing a synchronization.

Prototypes

' Visual Basic
Public Property **DisableConcurrency** As Boolean

// C#
public bool **DisableConcurrency** {get;set;}

Property value

True to disable concurrent access to UltraLite while performing a synchronization, false to enable concurrent access. The default is false.

Remarks

By default, other threads may perform UltraLite operations while a thread is synchronizing. When concurrent synchronization is disabled, other threads

will block on UltraLite calls until synchronization completes.

DownloadOnly property

Specifies whether to disable or enable uploads when synchronizing.

Prototypes

' Visual Basic
Public Property **DownloadOnly** As Boolean

// C#
public bool **DownloadOnly** {get;set;}

Property value

True to disable uploads when synchronizing, false to enable uploads. The default is false.

Remarks

At most, only one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) may be specified at a time. If more than one of these parameters is set to true, a ULSQLCode enumeration SQLException is thrown by Synchronize method.

See also

- "ULSyncParms class" on page 263
- "ULSyncParms members" on page 263
- "UploadOnly property" on page 271

KeepPartialDownload property

Specifies whether to disable or enable partial downloads when synchronizing.

Prototypes

' Visual Basic
Public Property **KeepPartialDownload** As Boolean

// C#
public bool **KeepPartialDownload** {get;set;}

Property value

True to enable partial downloads when synchronizing, false to disable partial downloads. The default is false.

Remarks

UltraLite.NET has the ability to restart downloads that fail because of communication errors or user aborts through the ULSyncProgressListener. UltraLite.NET processes the download as it is received. If a download is interrupted, then the partial download transaction remains in the database and can be resumed during the next synchronization.

To indicate that UltraLite.NET should save partial downloads, specify connection.SyncParms.KeepPartialDownload=true; otherwise the download is rolled back if an error occurs.

If a partial download was kept, then the output field connection.SyncResult.PartialDownloadRetained property is set to true when connection.Synchronize() exits.

If PartialDownloadRetained is set, then you can resume a download. To do this, call connection.Synchronize() with connection.SyncParms.ResumePartialDownload property set to true. It is recommended that you keep KeepPartialDownload set to true as well in case another communications error occurs. No upload is done if a download is skipped.

The download you receive during a resumed download will be as old as when the download originally began. If you need the most up to date data, then you can do another download immediately after the special resumed download completes.

When resuming a download, many of the ULSyncParms fields are not relevant. For example, the PublicationMask field is not used. You will receive the publications that you requested on the initial download. The only fields that need to be set are ResumePartialDownload property and UserName property. The fields KeepPartialDownload and DisableConcurrency property may be set if desired and will function as normal.

If you have a partial download and it is no longer needed, then you can call RollbackPartialDownload method to roll back the failed download transaction. Also, if you attempt to synchronize again and do not specify ResumePartialDownload, then the partial download is rolled back before the next synchronization begins.

For more information, refer to the "Resuming failed downloads" [*MobiLink Administration Guide*, page 74].

See also

- "ULSyncParms class" on page 263
- "ULSyncParms members" on page 263
- "PartialDownloadRetained property" on page 286
- "ResumePartialDownload property" on page 269
- "RollbackPartialDownload method" on page 95

NewPassword property

Specifies a new MobiLink password for the user specified with UserName.

Prototypes

' Visual Basic
Public Property **NewPassword** As String

// C#
public string **NewPassword** {get;set;}

Property value
A string specifying a new MobiLink password. The default is a null reference (Nothing in Visual Basic), meaning the password is not changed.

Remarks
A new password takes effect after the next synchronization.

See also
- "ULSyncParms class" on page 263
- "ULSyncParms members" on page 263
- "UserName property" on page 272

Password property

The MobiLink password for the user specified by UserName.

Prototypes

```
' Visual Basic
Public Property Password As String
```

```
// C#
public string  Password {get;set;}
```

Property value
A string specifying the MobiLink password. The default is a null reference (Nothing in Visual Basic), meaning no password is specified.

Remarks
The MobiLink user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink synchronization server.

See also
- "ULSyncParms class" on page 263
- "ULSyncParms members" on page 263
- "NewPassword property" on page 267
- "UserName property" on page 272

PingOnly property

Specifies whether the client should only ping the MobiLink synchronization server instead of performing a real synchronization.

Prototypes

```
' Visual Basic
Public Property PingOnly As Boolean
```

```
// C#
public bool PingOnly {get;set;}
```

Property value
True to specify that the client should only ping the MobiLink synchronization server, false to specify the client should perform a real synchronization. The default is false.

Remarks
At most, only one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) may be specified at a time. If more than one of these parameters is

set to true, a ULSQLCode enumeration SQLException is thrown by Synchronize method.

PublicationMask property

Specifies the publications to be synchronized.

Prototypes

' Visual Basic
Public Property **PublicationMask** As Integer

// C#
public int **PublicationMask** {get;set;}

Property value

A bitwise combination of publication masks, the special value SYNC_ALL_PUBS field, or the special value SYNC_ALL_DB field. The default is SYNC_ALL_DB field. See ULPublicationSchema class for more information on publication masks.

See also

- "ULSyncParms class" on page 263
- "ULSyncParms members" on page 263
- "Mask property" on page 227

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Prototypes

' Visual Basic
Public Property **ResumePartialDownload** As Boolean

// C#
public bool **ResumePartialDownload** {get;set;}

Property value

True to resume a previous partial download, false to discard a previous partial download. The default is false.

Remarks

Only at most one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) may be specified at a time. If more than one of these parameters is set to true, a ULSQLCode enumeration SQLException is thrown by Synchronize method.

For more information on partial downloads, see KeepPartialDownload property.

See also

- "ULSyncParms class" on page 263
- "ULSyncParms members" on page 263
- "PartialDownloadRetained property" on page 286

SendColumnNames property

Specifies whether the client should send column names to the MobiLink synchronization server during synchronization.

Prototypes

' Visual Basic
Public Property **SendColumnNames** As Boolean

// C#
public bool **SendColumnNames** {get;set;}

Property value

True to specify that the client should send column names to the MobiLink synchronization server, false to specify that column names are not sent. The default is false.

Remarks

This parameter is typically used together with the -za or -ze option on the MobiLink synchronization server for automatically generating synchronization scripts.

SendDownloadAck property

Specifies whether the client should send a download acknowledgement to the MobiLink synchronization server during synchronization.

Prototypes

' Visual Basic
Public Property **SendDownloadAck** As Boolean

// C#
public bool **SendDownloadAck** {get;set;}

Property value

True to specify that the client should send a download acknowledgement to the MobiLink synchronization server, false to specify that no download acknowledgement is sent. The default is false.

Remarks

If the client sends a download acknowledgement, the MobiLink synchronization server worker thread must wait for the client to apply the download. If the client does not sent a download acknowledgement, the MobiLink synchronization server is freed up sooner for its next synchronization.

Stream property

Specifies the MobiLink synchronization stream to use for synchronization.

Prototypes

' Visual Basic
Public Property **Stream** As ULStreamType

// C#
public ULStreamType **Stream** {get;set;}

Property value

One of the ULStreamType enumeration values specifying the type of synchronization stream to use. This parameter has no default value, and must be explicitly set.

Remarks

Most synchronization streams require parameters to identify the MobiLink synchronization server address and control other behavior. These parameters are supplied by the StreamParms property property.

If the stream type is set to a value that is invalid for the platform, the stream type will be set to ULStreamType enumeration.

See also

- "ULSyncParms class" on page 263
- "ULSyncParms members" on page 263
- "ULStreamType enumeration" on page 262
- "StreamParms property" on page 271

StreamParms property

Specifies the parameters to configure the synchronization stream.

Prototypes

' Visual Basic
Public Property **StreamParms** As String

// C#
public string **StreamParms** {get;set;}

Property value

A string, in the form of a semicolon-separated list of keyword-value pairs, specifying the parameters for the stream. The default is a null reference (Nothing in Visual Basic).

Remarks

For information on configuring specific stream types, refer to "Network protocol options for UltraLite synchronization clients" [*MobiLink Clients*, page 341].

StreamParms is a string containing all the parameters used for synchronization streams. Parameters are specified as a semicolon-separated list of name=value pairs ("param1=value1;param2=value2").

See also

- "ULSyncParms class" on page 263
- "ULSyncParms members" on page 263
- "Stream property" on page 270
- "ULStreamType enumeration" on page 262

UploadOnly property

Specifies whether to disable or enable downloads when synchronizing.

Prototypes

' Visual Basic
Public Property **UploadOnly** As Boolean

// C#
public bool **UploadOnly** {get;set;}

Property value

True to disable downloads, false to enable downloads. The default is false.

Remarks

At most, only one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) may be specified at a time. If more than one of these parameters is set to true, a ULSQLCode enumeration SQLException is thrown by Synchronize method.

See also

- "ULSyncParms class" on page 263
- "ULSyncParms members" on page 263
- "DownloadOnly property" on page 266

UserName property

The user name that uniquely identifies the MobiLink client to the MobiLink synchronization server.

Prototypes

' Visual Basic
Public Property **UserName** As String

// C#
public string **UserName** {get;set;}

Property value

A string specifying the user name. This parameter has no default value, and must be explicitly set.

Remarks

MobiLink uses this value to determine the download content, to record the synchronization state, and to recover from interruptions during synchronization. This user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink synchronization server.

See also

- "ULSyncParms class" on page 263
- "ULSyncParms members" on page 263
- "Password property" on page 268

Version property

Specifies which synchronization script to use.

Prototypes

' Visual Basic
Public Property **Version** As String

```
// C#
public string  Version {get;set;}
```

Property value

A string specifying the version of the synchronization script to use. This parameter has no default value, and must be explicitly set.

Remarks

Each synchronization script in the consolidated database is marked with a version string. For example, there may be two different download_cursor scripts, with each one identified by different a version string. The version string allows an UltraLite application to choose from a set of synchronization scripts.

CopyFrom method

Copies the properties of the specified ULSyncParms class object to this ULSyncParms class object.

Prototypes

```
' Visual Basic
Public Sub CopyFrom( _
  ByVal src As ULSyncParms _
)
```

```
// C#
public void CopyFrom(
  ULSyncParms src
);
```

Parameters

- **src** The object to copy from.

Finalize method

Releases unmanaged resources and performs other cleanup operations before the ULSyncParms is reclaimed by garbage collection.

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()
```

```
// C#
protected override void Finalize();
```

ULSyncProgressData class

UL Ext.: Returns synchronization progress monitoring data.

Prototypes

' Visual Basic
Public Class **ULSyncProgressData**

// C#
public class **ULSyncProgressData**

See also

- "ULSyncProgressData members" on page 274
- "ULSyncProgressListener interface" on page 281

ULSyncProgressData members

Public instance properties

Member	Description
ErrorMessage property	Returns the error message describing the error that occurred during synchronization.
ReceivedBytes property	Returns the number of bytes received so far. This information is updated for all states.
ReceivedDeletes property	Returns the number of deleted rows received so far.
ReceivedInserts property	Returns the number of inserted rows received so far.
ReceivedUpdates property	Returns the number of updated rows received so far.
SentBytes property	Returns the number of bytes sent so far. This information is updated for all states.
SentDeletes property	Returns the number of deleted rows sent so far.
SentInserts property	Returns the number of inserted rows sent so far.
SentUpdates property	Returns the number of updated rows sent so far.
SQLCode property	Returns the SQL code for synchronization.
State property	Returns the current synchronization state.
SyncParms property	Returns a reference to the connection's SyncParms object.
SyncResult property	Returns a reference to the connection's SyncResult object. This object is only updated with ULSyncProgressState enumeration and ULSyncProgressState enumeration events.
TableCount property	A count of the tables sent or received (TableCount of TableTotal) so far.

Member	Description
TableIndex property	Returns the index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).
TableTotal property	Returns the number of tables being synchronized.

ErrorMessage property

Returns the error message describing the error that occurred during synchronization.

Prototypes

```
' Visual Basic
Public Readonly Property ErrorMessage As String

// C#
public string  ErrorMessage {get;}
```

Property value

A string describing the error that occurred during synchronization.

See also

- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282

ReceivedBytes property

Returns the number of bytes received so far. This information is updated for all states.

Prototypes

```
' Visual Basic
Public Readonly Property ReceivedBytes As Long

// C#
public long ReceivedBytes {get;}
```

Property value

The number of bytes received so far.

See also

- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282
- "ULSyncProgressState enumeration" on page 282

ReceivedDeletes property

Returns the number of deleted rows received so far.

Prototypes

```
' Visual Basic
Public Readonly Property ReceivedDeletes As Integer
```

// C#
public int **ReceivedDeletes** {get;}

Property value — The number of deleted rows received so far.

See also
- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282
- "ULSyncProgressState enumeration" on page 282

ReceivedInserts property

Returns the number of inserted rows received so far.

Prototypes

' Visual Basic
Public Readonly Property **ReceivedInserts** As Integer

// C#
public int **ReceivedInserts** {get;}

Property value — The number of inserted rows received so far.

See also
- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282
- "ULSyncProgressState enumeration" on page 282

ReceivedUpdates property

Returns the number of updated rows received so far.

Prototypes

' Visual Basic
Public Readonly Property **ReceivedUpdates** As Integer

// C#
public int **ReceivedUpdates** {get;}

Property value — The number of updated rows received so far.

See also
- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282
- "ULSyncProgressState enumeration" on page 282

SentBytes property

Returns the number of bytes sent so far. This information is updated for all states.

Prototypes

' Visual Basic
Public Readonly Property **SentBytes** As Long

// C#
public long **SentBytes** {get;}

Property value

The number of bytes sent so far.

See also

- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282
- "ULSyncProgressState enumeration" on page 282

SentDeletes property

Returns the number of deleted rows sent so far.

Prototypes

' Visual Basic
Public Readonly Property **SentDeletes** As Integer

// C#
public int **SentDeletes** {get;}

Property value

The number of deleted rows sent so far.

See also

- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282
- "ULSyncProgressState enumeration" on page 282

SentInserts property

Returns the number of inserted rows sent so far.

Prototypes

' Visual Basic
Public Readonly Property **SentInserts** As Integer

// C#
public int **SentInserts** {get;}

Property value

The number of inserted rows sent so far.

See also

- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282
- "ULSyncProgressState enumeration" on page 282

SentUpdates property

Returns the number of updated rows sent so far.

Prototypes

' Visual Basic
Public Readonly Property **SentUpdates** As Integer

// C#
public int **SentUpdates** {get;}

Property value

The number of updated rows sent so far.

See also

- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282
- "ULSyncProgressState enumeration" on page 282

SQLCode property

Returns the SQL code for synchronization.

Prototypes

' Visual Basic
Public Readonly Property **SQLCode** As ULSQLCode

// C#
public ULSQLCode **SQLCode** {get;}

Property value

The ULSQLCode enumeration value for any synchronization error.

See also

- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282

State property

Returns the current synchronization state.

Prototypes

' Visual Basic
Public Readonly Property **State** As ULSyncProgressState

// C#
public ULSyncProgressState **State** {get;}

Property value

One of the ULSyncProgressState enumeration values specifying the current synchronization state.

See also

- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282

SyncParms property

Returns a reference to the connection's SyncParms object.

Prototypes

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

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

Property value

A reference to the SyncParms property object.

See also

- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282
- "ULSyncProgressState enumeration" on page 282
- "SyncParms property" on page 81

SyncResult property

Returns a reference to the connection's SyncResult object. This object is only updated with ULSyncProgressState enumeration and ULSyncProgressState enumeration events.

Prototypes

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

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

Property value

A reference to the SyncResult property object.

See also

- "ULSyncProgressData class" on page 274
- "ULSyncProgressData members" on page 274
- "ULSyncProgressState enumeration" on page 282
- "ULSyncProgressState enumeration" on page 282

TableCount property

A count of the tables sent or received (TableCount of TableTotal) so far.

Prototypes

' Visual Basic
Public Readonly Property **TableCount** As Integer

// C#
public int **TableCount** {get;}

Property value

The count of tables sent or received.

See also

- “ULSyncProgressData class” on page 274
- “ULSyncProgressData members” on page 274
- “ULSyncProgressState enumeration” on page 282
- “ULSyncProgressState enumeration” on page 282

TableIndex property

Returns the index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).

Prototypes

’ Visual Basic
Public Readonly Property **TableIndex** As Integer

// C#
public int **TableIndex** {get;}

Property value

The index of the table currently being synchronized. Tables are numbered 1 to TableCount property.

See also

- “ULSyncProgressData class” on page 274
- “ULSyncProgressData members” on page 274
- “ULSyncProgressState enumeration” on page 282
- “ULSyncProgressState enumeration” on page 282

TableTotal property

Returns the number of tables being synchronized.

Prototypes

’ Visual Basic
Public Readonly Property **TableTotal** As Integer

// C#
public int **TableTotal** {get;}

Property value

The number of tables being synchronized.

See also

- “ULSyncProgressData class” on page 274
- “ULSyncProgressData members” on page 274
- “ULSyncProgressState enumeration” on page 282
- “ULSyncProgressState enumeration” on page 282

ULSyncProgressListener interface

UL Ext.: The listener interface for receiving synchronization progress events.

Prototypes

' Visual Basic
Public Interface **ULSyncProgressListener**

// C#
public interface **ULSyncProgressListener**

See also

- "ULSyncProgressListener members" on page 281
- "Synchronize method" on page 96

ULSyncProgressListener members

Public instance methods

Member	Description
SyncProgressed method	Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue.

SyncProgressed method

Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue.

Prototypes

' Visual Basic
Public Function **SyncProgressed(** _
  ByVal *data* As ULSyncProgressData _
) As Boolean

// C#
public bool **SyncProgressed(**
  ULSyncProgressData *data*
);

Parameters

- **data** A ULSyncProgressData class object containing the latest synchronization progress data.

Return value

This method should return true to cancel synchronization or return false to continue.

Remarks

No UltraLite.NET API methods should be invoked during a SyncProgressed call.

ULSyncProgressState enumeration

UL Ext.: Enumerates all the states that can occur while synchronizing.

Prototypes

' Visual Basic
Public Enum **ULSyncProgressState**

// C#
public enum **ULSyncProgressState**

Members

Member	Description
STATE_CANCELLED	Synchronization has been cancelled.
STATE_COMMITTING_-DOWNLOAD	The download is being committed. The final count of rows received is included with this event. See ReceivedBytes property, ReceivedInserts property, ReceivedUpdates property, and ReceivedDeletes property.
STATE_CONNECTING	The synchronization stream has been built, but is not yet opened.
STATE_DISCONNECTING	The synchronization stream is about to be closed.
STATE_DONE	Synchronization has successfully completed. The connection's SyncResult property object has been updated.
STATE_ERROR	Synchronization has completed, but an error occurred. Check SyncResult property, ErrorMessage property, and SQLCode property for details.
STATE_FINISHING_UPLOAD	The upload is completing. The final count of rows sent is included with this event. See SentBytes property, SentInserts property, SentUpdates property, and SentDeletes property.
STATE_LAST	The last state entered by synchronization. This state is always entered and always after any other state. Other ULSyncProgressData fields may not contain valid data.
STATE_RECEIVING_DATA	Data for the current table is being received. ReceivedBytes property, ReceivedInserts property, ReceivedUpdates property, and ReceivedDeletes property have been updated.
STATE_RECEIVING_TABLE	A table is being received. Progress can be monitored using TableIndex property and TableCount property.
STATE_RECEIVING_-UPLOAD_ACK	An acknowledgement that the upload is complete is being received.

Member	Description
STATE_ROLLING_BACK_-DOWNLOAD	Synchronization is rolling back the download because an error was encountered during the download. The error will be reported with a subsequent STATE_ERROR progress report.
STATE_SENDING_DATA	Data for the current table is being sent. SentBytes property, SentInserts property, SentUpdates property, and SentDeletes property have been updated.
STATE_SENDING_-DOWNLOAD_ACK	An acknowledgement that the download is complete is being sent.
STATE_SENDING_HEADER	The synchronization stream has been opened and the header is about to be sent.
STATE_SENDING_TABLE	A table is being sent. Progress can be monitored using TableIndex property and TableCount property.
STATE_STARTING	No synchronization actions have been taken yet.

See also

- "ULSyncProgressData class" on page 274

ULSyncResult class

UL Ext.: Represents the status of the last synchronization.

Prototypes

' Visual Basic
Public Class **ULSyncResult**

// C#
public class **ULSyncResult**

Remarks

This class cannot be directly instantiated. Each connection has its own ULSyncResult instance, attached as its SyncResult property property. A ULSyncResult instance is only valid while that connection is open.

See also

- "ULSyncResult members" on page 284
- "SyncResult property" on page 82
- "Synchronize method" on page 96

ULSyncResult members

Public instance properties

Member	Description
AuthStatus property	Returns the authorization status code for the last synchronization attempt.
AuthValue property	Returns the return value from custom user authentication synchronization scripts.
IgnoredRows property	Checks whether any uploaded rows were ignored or not during the last synchronization.
PartialDownloadRetained property	Checks whether a partial download was retained or not during the last synchronization.
StreamErrorCode property	Returns the error reported by the stream itself.
StreamErrorContext property	Returns the basic network operation being performed when the stream error occurred.
StreamErrorID property	Returns the ID of the network layer reporting an error.
StreamErrorSystem property	Returns the stream error system-specific code.
Timestamp property	Returns the timestamp of the last synchronization.
UploadOK property	Checks whether the last upload synchronization was successful or not.

Protected instance methods

Member	Description
Finalize method	Releases unmanaged resources and performs other cleanup operations before the ULSyncResult is reclaimed by garbage collection.

AuthStatus property

Returns the authorization status code for the last synchronization attempt.

Prototypes

' Visual Basic
Public Readonly Property **AuthStatus** As ULAuthStatusCode

// C#
public ULAuthStatusCode **AuthStatus** {get;}

Property value

One of the ULAuthStatusCode enumeration values denoting the authorization status for the last synchronization attempt.

AuthValue property

Returns the return value from custom user authentication synchronization scripts.

Prototypes

' Visual Basic
Public Readonly Property **AuthValue** As Long

// C#
public long **AuthValue** {get;}

Property value

A long integer returned from custom user authentication synchronization scripts.

IgnoredRows property

Checks whether any uploaded rows were ignored or not during the last synchronization.

Prototypes

' Visual Basic
Public Readonly Property **IgnoredRows** As Boolean

// C#
public bool **IgnoredRows** {get;}

Property value

True if any uploaded rows were ignored during the last synchronization, false if no rows were ignored.

See also

- "ULSyncResult class" on page 284
- "ULSyncResult members" on page 284

- “DownloadOnly property” on page 266

PartialDownloadRetained property

Checks whether a partial download was retained or not during the last synchronization.

Prototypes

’ Visual Basic
Public Readonly Property **PartialDownloadRetained** As Boolean

// C#
public bool **PartialDownloadRetained** {get;}

Property value

True if a download was interrupted and the partial download was retained, false if the download was not interrupted or if the partial download was rolled back.

See also

- “ULSyncResult class” on page 284
- “ULSyncResult members” on page 284
- “KeepPartialDownload property” on page 266

StreamErrorCode property

Returns the error reported by the stream itself.

Prototypes

’ Visual Basic
Public Readonly Property **StreamErrorCode** As ULStreamErrorCode

// C#
public ULStreamErrorCode **StreamErrorCode** {get;}

Property value

One of the ULStreamErrorCode enumeration values denoting the error reported by the stream itself, ULStreamErrorCode enumeration if no error occurred.

StreamErrorContext property

Returns the basic network operation being performed when the stream error occurred.

Prototypes

’ Visual Basic
Public Readonly Property **StreamErrorContext** As ULStreamErrorContext

// C#
public ULStreamErrorContext **StreamErrorContext** {get;}

Property value

One of the ULStreamErrorContext enumeration values denoting which basic network operation was being performed when the stream error occurred.

StreamErrorID property

Returns the ID of the network layer reporting an error.

Prototypes

' Visual Basic
Public Readonly Property **StreamErrorID** As ULStreamErrorID

// C#
public ULStreamErrorID **StreamErrorID** {get;}

Property value

One of the ULStreamErrorID enumeration values denoting the ID of the network layer reporting an error.

StreamErrorSystem property

Returns the stream error system-specific code.

Prototypes

' Visual Basic
Public Readonly Property **StreamErrorSystem** As Integer

// C#
public int **StreamErrorSystem** {get;}

Property value

An integer denoting the stream error system-specific code.

Timestamp property

Returns the timestamp of the last synchronization.

Prototypes

' Visual Basic
Public Readonly Property **Timestamp** As Date

// C#
public DateTime **Timestamp** {get;}

Property value

A DateTime specifying the timestamp of the last synchronization.

UploadOK property

Checks whether the last upload synchronization was successful or not.

Prototypes

' Visual Basic
Public Readonly Property **UploadOK** As Boolean

// C#
public bool **UploadOK** {get;}

Property value

True if the last upload synchronization was successful, false if the last upload synchronization was unsuccessful.

Finalize method

Releases unmanaged resources and performs other cleanup operations before the ULSyncResult is reclaimed by garbage collection.

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

ULTable class

UL Ext.: Represents a table in an UltraLite database.

Prototypes

```
' Visual Basic
Public Class ULTable
  Inherits ULDataReader

// C#
public class ULTable :
  ULDataReader
```

Remarks

This class cannot be directly instantiated. Tables are created using the ExecuteTable method method of the ULCommand class class.

ULTable members

Public instance properties

Member	Description
Depth property (inherited from ULDataReader)	Returns the depth of nesting for the current row. The outermost table has a depth of zero.
FieldCount property (inherited from ULDataReader)	Returns the number of columns in the cursor.
IsBOF property (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is before the first row or not.
IsClosed property (inherited from ULDataReader)	Checks whether the cursor is currently open.
IsEOF property (inherited from ULDataReader)	**UL Ext.:** Checks whether the current row position is after the last row or not.
Item property (inherited from ULDataReader)	Returns the value of the specified column in its native format. In C#, this property is the indexer for the ULDataReader class.
Item property (inherited from ULDataReader)	Returns the value of the specified named column in its native format. In C#, this property is the indexer for the ULDataReader class.
RecordsAffected property (inherited from ULDataReader)	Returns the number of rows changed, inserted, or deleted by execution of the SQL statement. For SELECT statements or CommandType.TableDirect tables, this value is -1.
RowCount property (inherited from ULDataReader)	**UL Ext.:** Returns the number of rows in the cursor.
Schema property	Holds the table schema. This property is only valid while its connection is open.

Public instance methods

Member	Description
AppendBytes method	Appends the specified subset of the specified array of Bytes to the new value for the specified ULDbType enumeration column.
AppendChars method	Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType enumeration column.
Close method (inherited from ULDataReader)	Closes the cursor.
Delete method	Deletes the current row.
DeleteAllRows method	Deletes all rows in the table.
Dispose method (inherited from ULDataReader)	Releases the unmanaged resources used by the ULDataReader and optionally releases the managed resources.
FindBegin method	Prepares to perform a new Find on a table.
FindFirst method	Moves forwards through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.
FindFirst method	Moves forwards through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.
FindLast method	Moves backwards through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.
FindLast method	Moves backwards through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.
FindNext method	Continues a FindFirst method search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.
FindNext method	Continues a FindFirst method search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or partial set of values in the current index.
FindPrevious method	Continues a FindLast method search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.

Member	Description
FindPrevious method	Continues a FindLast method search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or partial set of values in the current index.
GetBoolean method (inherited from ULDataReader)	Returns the value for the specified column as a Boolean.
GetByte method (inherited from ULDataReader)	Returns the value for the specified column as an unsigned 8-bit value (Byte).
GetBytes method (inherited from ULDataReader)	Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination Byte array.
GetBytes method (inherited from ULDataReader)	**UL Ext.:** Returns the value for the specified column as an array of Bytes. Only valid for columns of type ULDbType enumeration, ULDbType enumeration, or ULDbType enumeration.
GetChar method (inherited from ULDataReader)	This method is not supported in UltraLite.NET.
GetChars method (inherited from ULDataReader)	Copies a subset of the value for the specified ULDbType enumeration column, beginning at the specified offset, to the specified offset of the destination System.Char array.
GetData method (inherited from ULDataReader)	This method is not supported in UltraLite.NET.
GetDataTypeName method (inherited from ULDataReader)	Returns the name of the specified column's provider data type.
GetDateTime method (inherited from ULDataReader)	Returns the value for the specified column as a DateTime with millisecond accuracy.
GetDecimal method (inherited from ULDataReader)	Returns the value for the specified column as a Decimal.
GetDouble method (inherited from ULDataReader)	Returns the value for the specified column as a Double.
GetFieldType method (inherited from ULDataReader)	Returns the Type most appropriate for the specified column.
GetFloat method (inherited from ULDataReader)	Returns the value for the specified column as a Single.
GetGuid method (inherited from ULDataReader)	Returns the value for the specified column as a UUID (Guid).

Member	Description
GetInt16 method (inherited from ULDataReader)	Returns the value for the specified column as an Int16.
GetInt32 method (inherited from ULDataReader)	Returns the value for the specified column as an Int32.
GetInt64 method (inherited from ULDataReader)	Returns the value for the specified column as an Int64.
GetName method (inherited from ULDataReader)	Returns the name of the specified column.
GetOrdinal method (inherited from ULDataReader)	Returns the column ID of the named column.
GetSchemaTable method (inherited from ULDataReader)	Returns a DataTable that describes the column metadata of the ULDataReader.
GetString method (inherited from ULDataReader)	Returns the value for the specified column as a String.
GetTimeSpan method (inherited from ULDataReader)	Returns the value for the specified column as a TimeSpan with millisecond accuracy.
GetUInt16 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt16.
GetUInt32 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt32.
GetUInt64 method (inherited from ULDataReader)	Returns the value for the specified column as a UInt64.
GetValue method (inherited from ULDataReader)	Returns the value of the specified column in its native format.
GetValues method (inherited from ULDataReader)	Returns all the column values for the current row.
Insert method	Inserts a new row with the current column values (specified using the set methods). Each insert must be preceded by a call to InsertBegin method.
InsertBegin method	Prepares to insert a new row into the table by setting all current column values to their default values.
IsDBNull method (inherited from ULDataReader)	Checks whether the value from the specified column is NULL.

Member	Description
LookupBackward method	Moves backwards through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.
LookupBackward method	Moves backwards through the table from the beginning, looking for a row that matches or is less than a value or partial set of values in the current index.
LookupBegin method	Prepares to perform a new lookup on the table. The value(s) to search for are specified by calling the appropriate setType method(s) on the columns in the index the table was opened with.
LookupForward method	Moves forwards through the table from the beginning, looking for a row that matches or is greater than a value or full set of values in the current index.
LookupForward method	Moves forwards through the table from the beginning, looking for a row that matches or is greater than a value or partial set of values in the current index.
MoveAfterLast method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to after the last row of the cursor.
MoveBeforeFirst method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to before the first row of the cursor.
MoveFirst method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the first row of the cursor.
MoveLast method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the last row of the cursor.
MoveNext method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the next row or after the last row if the cursor was already on the last row.
MovePrevious method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor to the previous row or before the first row.
MoveRelative method (inherited from ULDataReader)	**UL Ext.:** Positions the cursor relative to the current row.
NextResult method (inherited from ULDataReader)	Advances the ULDataReader to the next result when reading the results of batch SQL statements.
Read method (inherited from ULDataReader)	Positions the cursor to the next row, or after the last row if the cursor was already on the last row.
SetBoolean method	Sets the value for the specified column using a Boolean.

Member	Description
SetByte method	Sets the value for the specified column using a Byte (unsigned 8-bit integer).
SetBytes method	Sets the value for the specified column using an array of Bytes.
SetDateTime method	Sets the value for the specified column using a DateTime.
SetDBNull method	Sets a column to NULL.
SetDecimal method	Sets the value for the specified column using a Decimal.
SetDouble method	Sets the value for the specified column using a Double.
SetFloat method	Sets the value for the specified column using a Single.
SetGuid method	Sets the value for the specified column using a Guid.
SetInt16 method	Sets the value for the specified column using an Int16.
SetInt32 method	Sets the value for the specified column using an Int32.
SetInt64 method	Sets the value for the specified column using an Int64.
SetString method	Sets the value for the specified column using a String.
SetTimeSpan method	Sets the value for the specified column using a TimeSpan.
SetToDefault method	Sets the value for the specified column to its default value.
SetUInt16 method	Sets the value for the specified column using a UInt16.
SetUInt32 method	Sets the value for the specified column using an UInt32.
SetUInt64 method	Sets the value for the specified column using a UInt64.
Truncate method	Deletes all rows in the table while temporarily activating a stop synchronization delete.
Update method	Updates the current row with the current column values (specified using the set methods). Each update must be preceded by a call to UpdateBegin method.
UpdateBegin method	Prepares to update the current row in the table.

Protected instance methods

Member	Description
Finalize method	Releases unmanaged resources and performs other cleanup operations before the ULTable is reclaimed by garbage collection.

Schema property

Holds the table schema. This property is only valid while its connection is open.

Prototypes

```
' Visual Basic
Public Readonly Property Schema As ULTableSchema
```

```
// C#
public ULTableSchema Schema {get;}
```

Property value

The ULTableSchema class object representing the table schema.

Remarks

This property represents the complete schema of the table, including UltraLite.NET extended information which is not represented in the results from GetSchemaTable method.

AppendBytes method

Appends the specified subset of the specified array of Bytes to the new value for the specified ULDbType enumeration column.

Prototypes

```
' Visual Basic
Public Sub AppendBytes( _
  ByVal columnID As Integer, _
  ByVal val As Byte(), _
  ByVal srcOffset As Integer, _
  ByVal count As Integer _
)
```

```
// C#
public void AppendBytes(
  int columnID,
  byte[] val,
  int srcOffset,
  int count
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.
- **val** The value to append to the current new value for the column.
- **srcOffset** The start position in the source array.
- **count** The number of bytes to be copied.

Remarks

The bytes at position *srcOffset* (starting from 0) through *srcOffset*+*count*-1 of the array *val* are appended to the value for the specified column.

When inserting, InsertBegin method initializes the new value to the column's default value. The data in the row is not actually changed until you execute an Insert method, and that change is not made permanent until it is committed.

When updating, the first append on a column will clear the current value prior to appending the new value.

If any of the following are true, a ULException class with code ULSQLCode enumeration is thrown and the destination is not modified:

- *val* is null.
- *srcOffset* is negative.
- *count* is negative.
- *srcOffset*+*count* is greater than *val*.Length.

For other errors, a ULException class with the appropriate error code is thrown.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166

AppendChars method

Appends the specified subset of the specified array of System.Chars to the new value for the specified ULDbType enumeration column.

Prototypes

```
' Visual Basic
Public Sub AppendChars( _
  ByVal columnID As Integer, _
  ByVal val As Char(), _
  ByVal srcOffset As Integer, _
  ByVal count As Integer _
)
```

```
// C#
public void AppendChars(
  int columnID,
  char[] val,
  int srcOffset,
  int count
);
```

Parameters

- **columnID** The ID number of the column. The value must be in the range [0,FieldCount property-1]. The first column in the cursor has an ID value of zero.
- **val** The value to append to the current new value for the column.
- **srcOffset** The start position in the source array.
- **count** The number of bytes to be copied.

Remarks

The characters at position *srcOffset* (starting from 0) through *srcOffset*+*count*-1 of the array *val* are appended to the value for the specified column. When inserting, InsertBegin method initializes the new value to the column's default value. The data in the row is not actually changed until you execute an Insert method, and that change is not made permanent until it is committed.

When updating, the first append on a column clears the current value prior to appending the new value.

If any of the following is true, a ULException class with code ULSQLCode enumeration is thrown and the destination is not modified:

- *val* is null.
- *srcOffset* is negative.
- *count* is negative.
- *srcOffset*+*count* is greater than *value*.Length.

For other errors, a ULException class with the appropriate error code is thrown.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295

- "GetFieldType method" on page 166

Delete method

Deletes the current row.

Prototypes

' Visual Basic
Public Sub **Delete()**

// C#
public void **Delete();**

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "StartSynchronizationDelete method" on page 95
- "StopSynchronizationDelete method" on page 96

DeleteAllRows method

Deletes all rows in the table.

Prototypes

' Visual Basic
Public Sub **DeleteAllRows()**

// C#
public void **DeleteAllRows();**

Remarks

In some applications, it can be useful to delete all rows from a table before downloading a new set of data into the table. Rows can be deleted from the UltraLite database without being deleted from the consolidated database using StopSynchronizationDelete method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "Truncate method" on page 322

Finalize method

Releases unmanaged resources and performs other cleanup operations before the ULTable is reclaimed by garbage collection.

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

FindBegin method

Prepares to perform a new Find on a table.

Prototypes

' Visual Basic
Public Sub **FindBegin()**

// C#
public void **FindBegin();**

Remarks

The value(s) to search for are specified by calling the appropriate setType method(s) on the columns in the index the table was opened with.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "FindFirst method" on page 299
- "FindFirst method" on page 300
- "FindLast method" on page 300
- "FindLast method" on page 301

FindFirst method

Moves forwards through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.

Prototypes

' Visual Basic
Overloads Public Function **FindFirst()** As Boolean

// C#
public bool **FindFirst();**

Return value

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure, the cursor position is after the last row (IsEOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "FindBegin method" on page 299
- "FindNext method" on page 302
- "FindPrevious method" on page 303
- "FindFirst method" on page 300

FindFirst method

Moves forwards through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindFirst( _
  ByVal numColumns As Short _
) As Boolean

// C#
public bool FindFirst(
  short numColumns
);
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the find. For example, if you have a three column index and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a value of 1.

Return value

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure, the cursor position is after the last row (IsEOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "FindBegin method" on page 299
- "FindNext method" on page 302
- "FindPrevious method" on page 303
- "FindFirst method" on page 299

FindLast method

Moves backwards through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindLast() As Boolean

// C#
public bool FindLast();
```

Return value

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is before the first row (IsBOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "FindBegin method" on page 299
- "FindNext method" on page 302
- "FindPrevious method" on page 303
- "FindLast method" on page 301

FindLast method

Moves backwards through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindLast( _
  ByVal numColumns As Short _
) As Boolean

// C#
public bool FindLast(
  short numColumns
);
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the find. For example, if you have a three column index and you want to find a value that matches based on the first column only, you should set the value for the first column, then supply a value of 1.

Return value

True if successful, false otherwise

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is before the first row (IsBOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "FindBegin method" on page 299

- "FindNext method" on page 302
- "FindPrevious method" on page 303
- "FindLast method" on page 300

FindNext method

Continues a FindFirst method search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindNext() As Boolean

// C#
public bool FindNext();
```

Return value

True if successful, false otherwise.

Remarks

The cursor is left on the next row if it exactly matches the index value. On failure, the cursor position is after the last row (IsEOF property).

FindNext behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "FindFirst method" on page 299
- "FindNext method" on page 302

FindNext method

Continues a FindFirst method search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindNext( _
  ByVal numColumns As Short _
) As Boolean

// C#
public bool FindNext(
  short numColumns
);
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the find. For example, if you have a three column index, and you want to

find a value that matches based on the first column only, you should set the value for the first column, and then supply a value of 1.

Return value

True if successful, false otherwise.

Remarks

The cursor is left on the next row if it exactly matches the index value. On failure, the cursor position is after the last row (IsEOF property).

FindNext behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "FindFirst method" on page 300
- "FindNext method" on page 302

FindPrevious method

Continues a FindLast method search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindPrevious() As Boolean

// C#
public bool FindPrevious();
```

Return value

True if successful, false otherwise.

Remarks

The cursor is left on the previous row if it exactly matches the index value. On failure, the cursor position is before the first row (IsBOF property).

FindPrevious behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "FindLast method" on page 300
- "FindPrevious method" on page 303

FindPrevious method

Continues a FindLast method search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindPrevious( _
  ByVal numColumns As Short _
) As Boolean

// C#
public bool FindPrevious(
  short numColumns
);
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the find. For example, if you have a three column index and you want to look up a value that matches based on the first column only, you should set the value for the first column, then supply a value of 1.

Return value

True if successful, false otherwise.

Remarks

The cursor is left on the previous row if it exactly matches the index value. On failure, the cursor position is before the first row (IsBOF property).

FindPrevious behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "FindLast method" on page 301
- "FindPrevious method" on page 303

Insert method

Inserts a new row with the current column values (specified using the set methods).

Each insert must be preceded by a call to InsertBegin method.

Prototypes

```
' Visual Basic
Public Sub Insert()

// C#
public void Insert();
```

Exceptions

- ULException class - A SQL error occurred.

InsertBegin method

Prepares to insert a new row into the table by setting all current column values to their default values.

Prototypes

' Visual Basic
Public Sub **InsertBegin()**

// C#
public void **InsertBegin();**

Remarks

Call the appropriate SetType or AppendType method(s) to specify the non-default values that are to be inserted.

The row is not actually inserted and the data in the row is not actually changed until you execute the Insert method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "Insert method" on page 304

LookupBackward method

Moves backwards through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.

Prototypes

' Visual Basic
Overloads Public Function **LookupBackward()** As Boolean

// C#
public bool **LookupBackward();**

Return value

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is less than the index value. On failure, (no rows less than the value being looked for) the cursor position is before the first row (IsBOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "LookupBegin method" on page 306
- "LookupBackward method" on page 305

LookupBackward method

Moves backwards through the table from the beginning, looking for a row that matches or is less than a value or partial set of values in the current

index.

Prototypes

```
' Visual Basic
Overloads Public Function LookupBackward( _
  ByVal numColumns As Short _
) As Boolean

// C#
public bool LookupBackward(
  short numColumns
);
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a value of 1.

Return value

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is less than the index value. On failure, (no rows less than the value being looked for) the cursor position is before the first row (IsBOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "LookupBegin method" on page 306
- "LookupBackward method" on page 305

LookupBegin method

Prepares to perform a new lookup on the table. The value(s) to search for are specified by calling the appropriate setType method(s) on the columns in the index the table was opened with.

Prototypes

```
' Visual Basic
Public Sub LookupBegin()

// C#
public void LookupBegin();
```

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "LookupForward method" on page 307

- "LookupForward method" on page 307
- "LookupBackward method" on page 305
- "LookupBackward method" on page 305

LookupForward method

Moves forwards through the table from the beginning, looking for a row that matches or is greater than a value or full set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function LookupForward() As Boolean
```

```
// C#
public bool LookupForward();
```

Return value

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure, (no rows greater than the value being looked for) the cursor position is after the last row (IsEOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "LookupBegin method" on page 306
- "LookupForward method" on page 307

LookupForward method

Moves forwards through the table from the beginning, looking for a row that matches or is greater than a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function LookupForward( _
  ByVal numColumns As Short _
) As Boolean
```

```
// C#
public bool LookupForward(
  short numColumns
);
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a value of 1.

Return value

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure, (no rows greater than the value being looked for) the cursor position is after the last row (IsEOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "LookupBegin method" on page 306
- "LookupForward method" on page 307

SetBoolean method

Sets the value for the specified column using a Boolean.

Prototypes

```
' Visual Basic
Public Sub SetBoolean( _
  ByVal columnID As Integer, _
  ByVal val As Boolean _
)

// C#
public void SetBoolean(
  int columnID,
  bool val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304

- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166

SetByte method

Sets the value for the specified column using a Byte (unsigned 8-bit integer).

Prototypes

```
' Visual Basic
Public Sub SetByte( _
  ByVal columnID As Integer, _
  ByVal val As Byte _
)

// C#
public void SetByte(
  int columnID,
  byte val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166

SetBytes method

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

Prototypes

```
' Visual Basic
Public Sub SetBytes( _
  ByVal columnID As Integer, _
  ByVal val As Byte() _
)

// C#
public void SetBytes(
  int columnID,
  byte[] val
);
```

Parameters

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

Remarks

Only suitable for columns of type ULDbType enumeration or ULDbType enumeration, or for columns of type ULDbType enumeration when val is of length 16. The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULTable class” on page 289
- “ULTable members” on page 289
- “GetOrdinal method” on page 171
- “FindBegin method” on page 299
- “LookupBegin method” on page 306
- “InsertBegin method” on page 304
- “UpdateBegin method” on page 323
- “Schema property” on page 295
- “GetFieldType method” on page 166

SetDateTime method

Sets the value for the specified column using a DateTime.

Prototypes

```
' Visual Basic
Public Sub SetDateTime( _
  ByVal columnID As Integer, _
  ByVal val As Date _
)
```

```
// C#
public void SetDateTime(
  int columnID,
  DateTime val
);
```

Parameters

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

Remarks

The set value is accurate to the millisecond. The data in the row is not actually changed until you execute an Insert method or Update method, and that change is made not permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166

SetDBNull method

Sets a column to NULL.

Prototypes

```
' Visual Basic
Public Sub SetDBNull( _
  ByVal columnID As Integer _
)
```

```
// C#
public void SetDBNull(
  int columnID
);
```

Parameters

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

Remarks

The data is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULTable class” on page 289
- “ULTable members” on page 289
- “GetOrdinal method” on page 171
- “FindBegin method” on page 299
- “LookupBegin method” on page 306
- “InsertBegin method” on page 304
- “UpdateBegin method” on page 323
- “Schema property” on page 295
- “IsColumnNullable method” on page 333

SetDecimal method

Sets the value for the specified column using a Decimal.

Prototypes

```
' Visual Basic
Public Sub SetDecimal( _
  ByVal columnID As Integer, _
  ByVal val As Decimal _
)

// C#
public void SetDecimal(
  int columnID,
  decimal val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULTable class” on page 289
- “ULTable members” on page 289
- “GetOrdinal method” on page 171
- “FindBegin method” on page 299
- “LookupBegin method” on page 306
- “InsertBegin method” on page 304
- “UpdateBegin method” on page 323
- “Schema property” on page 295
- “GetFieldType method” on page 166

SetDouble method

Sets the value for the specified column using a Double.

Prototypes

```
' Visual Basic
Public Sub SetDouble( _
  ByVal columnID As Integer, _
  ByVal val As Double _
)

// C#
public void SetDouble(
  int columnID,
  double val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULTable class” on page 289
- “ULTable members” on page 289
- “GetOrdinal method” on page 171
- “FindBegin method” on page 299
- “LookupBegin method” on page 306
- “InsertBegin method” on page 304
- “UpdateBegin method” on page 323
- “Schema property” on page 295
- “GetFieldType method” on page 166

SetFloat method

Sets the value for the specified column using a Single.

Prototypes

```
' Visual Basic
Public Sub SetFloat( _
  ByVal columnID As Integer, _
  ByVal val As Single _
)
```

```
// C#
public void SetFloat(
  int columnID,
  float  val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166

SetGuid method

Sets the value for the specified column using a Guid.

Prototypes

```
' Visual Basic
Public Sub SetGuid( _
  ByVal columnID As Integer, _
  ByVal val As Guid _
)
```

```
// C#
public void SetGuid(
  int columnID,
  Guid val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed. Only valid for columns of type ULDbType enumeration or for columns of type ULDbType enumeration with length 16.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetNewUUID method" on page 92
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166
- "GetColumnSize method" on page 116

SetInt16 method

Sets the value for the specified column using an Int16.

Prototypes

```
' Visual Basic
Public Sub SetInt16( _
  ByVal columnID As Integer, _
  ByVal val As Short _
)

// C#
public void SetInt16(
  int columnID,
  short val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289

- “GetOrdinal method” on page 171
- “FindBegin method” on page 299
- “LookupBegin method” on page 306
- “InsertBegin method” on page 304
- “UpdateBegin method” on page 323
- “Schema property” on page 295
- “GetFieldType method” on page 166

SetInt32 method

Sets the value for the specified column using an Int32.

Prototypes

```
' Visual Basic
Public Sub SetInt32( _
  ByVal columnID As Integer, _
  ByVal val As Integer _
)

// C#
public void SetInt32(
  int columnID,
  int val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULTable class” on page 289
- “ULTable members” on page 289
- “GetOrdinal method” on page 171
- “FindBegin method” on page 299
- “LookupBegin method” on page 306
- “InsertBegin method” on page 304
- “UpdateBegin method” on page 323
- “Schema property” on page 295
- “GetFieldType method” on page 166

SetInt64 method

Sets the value for the specified column using an Int64.

Prototypes

```
' Visual Basic
Public Sub SetInt64( _
  ByVal columnID As Integer, _
  ByVal val As Long _
)

// C#
public void SetInt64(
  int columnID,
  long val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166

SetString method

Sets the value for the specified column using a String.

Prototypes

```
' Visual Basic
Public Sub SetString( _
  ByVal columnID As Integer, _
  ByVal val As String _
)

// C#
public void SetString(
  int columnID,
  string  val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166

SetTimeSpan method

Sets the value for the specified column using a TimeSpan.

Prototypes

```
' Visual Basic
Public Sub SetTimeSpan( _
  ByVal columnID As Integer, _
  ByVal val As TimeSpan _
)

// C#
public void SetTimeSpan(
  int columnID,
  TimeSpan val
);
```

Parameters

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

Remarks

The set value is accurate to the millisecond and is normalized to a nonnegative value between 0 and 24 hours. The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166

SetToDefault method

Sets the value for the specified column to its default value.

Prototypes

```
' Visual Basic
Public Sub SetToDefault( _
  ByVal columnID As Integer _
)

// C#
public void SetToDefault(
  int columnID
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166
- "GetColumnDefaultValue method" on page 327

SetUInt16 method

Sets the value for the specified column using a UInt16.

Prototypes

```
' Visual Basic
Public Sub SetUInt16( _
  ByVal columnID As Integer, _
  ByVal val As UInt16 _
)

// C#
public void SetUInt16(
  int columnID,
  ushort val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULTable class” on page 289
- “ULTable members” on page 289
- “GetOrdinal method” on page 171
- “FindBegin method” on page 299
- “LookupBegin method” on page 306
- “InsertBegin method” on page 304
- “UpdateBegin method” on page 323
- “Schema property” on page 295
- “GetFieldType method” on page 166

SetUInt32 method

Sets the value for the specified column using an UInt32.

Prototypes

```
' Visual Basic
Public Sub SetUInt32( _
  ByVal columnID As Integer, _
  ByVal val As UInt32 _
)
```

```
// C#
public void SetUInt32(
  int columnID,
  uint val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULTable class” on page 289
- “ULTable members” on page 289
- “GetOrdinal method” on page 171
- “FindBegin method” on page 299
- “LookupBegin method” on page 306
- “InsertBegin method” on page 304
- “UpdateBegin method” on page 323
- “Schema property” on page 295
- “GetFieldType method” on page 166

SetUInt64 method

Sets the value for the specified column using a UInt64.

Prototypes

```
' Visual Basic
Public Sub SetUInt64( _
  ByVal columnID As Integer, _
  ByVal val As UInt64 _
)
```

```
// C#
public void SetUInt64(
  int columnID,
  ulong val
);
```

Parameters

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

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not made permanent until it is committed.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "GetOrdinal method" on page 171
- "FindBegin method" on page 299
- "LookupBegin method" on page 306
- "InsertBegin method" on page 304
- "UpdateBegin method" on page 323
- "Schema property" on page 295
- "GetFieldType method" on page 166

Truncate method

Deletes all rows in the table while temporarily activating a stop synchronization delete.

Prototypes

```
' Visual Basic
Public Sub Truncate()

// C#
public void Truncate();
```

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTable class" on page 289
- "ULTable members" on page 289
- "DeleteAllRows method" on page 298

Update method

Updates the current row with the current column values (specified using the set methods).

Each update must be preceded by a call to UpdateBegin method.

Prototypes

```
' Visual Basic
Public Sub Update()

// C#
public void Update();
```

Exceptions

- ULException class - A SQL error occurred.

UpdateBegin method

Prepares to update the current row in the table.

Prototypes

' Visual Basic
Public Sub **UpdateBegin()**

// C#
public void **UpdateBegin();**

Remarks

Column values are modified by calling the appropriate setType or AppendType method(s). The first append on a column clears the current column value prior to appending the new value.

The data in the row is not actually changed until you call Update method, and that change is not made permanent until it is committed.

Modifying columns in the index used to open the table affects any active searches in unpredictable ways. Columns in the primary key of the table can not be updated.

Exceptions

- ULException class - A SQL error occurred.

ULTableSchema class

UL Ext.: Represents the schema of an UltraLite table.

Prototypes

```
' Visual Basic
NotInheritable Public Class ULTableSchema
  Inherits ULCursorSchema

// C#
public sealed class ULTableSchema :
  ULCursorSchema
```

Remarks

This class cannot be directly instantiated. A ULTableSchema class object is attached to a table as its Schema property property.

ULTableSchema members

Public instance properties

Member	Description
ColumnCount property (inherited from ULCursorSchema)	Returns the number of columns in the cursor.
IndexCount property	Returns the number of indexes on the table.
IsNeverSynchronized property	Checks whether the table is marked as never being synchronized.
IsOpen property (inherited from ULCursorSchema)	Checks whether the cursor schema is currently open.
Name property	Returns the name of the table.
PrimaryKey property	Returns the index schema of the primary key for the table.
UploadUnchangedRows property	Checks whether the database will upload rows that have not changed.

Public instance methods

Member	Description
GetColumnDefaultValue method	Returns the default value of the specified column.
GetColumnID method (inherited from ULCursorSchema)	Returns the column ID of the named column.
GetColumnName method (inherited from ULCursorSchema)	Returns the name of the column identified by the specified column ID.
GetColumnPartitionSize method	Returns the global autoincrement partition size assigned to the specified column.

Member	Description
GetColumnPrecision method (inherited from ULCursorSchema)	Returns the precision of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
GetColumnScale method (inherited from ULCursorSchema)	Returns the scale of the column identified by the specified column ID if the column is a numeric column (SQL type NUMERIC).
GetColumnSize method (inherited from ULCursorSchema)	Returns the size of the column identified by the specified column ID if the column is a sized column (SQL type BINARY or CHAR).
GetColumnULDbType method (inherited from ULCursorSchema)	Returns the UltraLite.NET data type of the column identified by the specified column ID.
GetIndex method	Returns the index schema of the named index.
GetIndexName method	Returns the name of the index identified by the specified index ID.
GetOptimalIndex method	The optimal index for searching a table using the specified column.
GetSchemaTable method (inherited from ULCursorSchema)	Returns a DataTable that describes the column schema of the ULDataReader class.
IsColumnAutoIncrement method	Checks whether the specified column's default is set to autoincrement.
IsColumnCurrentDate method	Checks whether the specified column's default is set to the current date (ULDbType enumeration).
IsColumnCurrentTime method	Checks whether the specified column's default is set to the current time (ULDbType enumeration).
IsColumnCurrentTimestamp method	Checks whether the specified column's default is set to the current timestamp (ULDbType enumeration).
IsColumnGlobalAutoIncrement method	Checks whether the specified column's default is set to global autoincrement.
IsColumnNewUUID method	Checks whether the specified column's default is set to a new UUID (Guid).
IsColumnNullable method	Checks whether the specified column is nullable.
IsInPublication method	Checks whether the table is contained in the named publication.

Protected instance methods

Member	Description
Finalize method	Releases unmanaged resources and performs other cleanup operations before the ULTableSchema is reclaimed by garbage collection.

IndexCount property

Returns the number of indexes on the table.

Prototypes

```
' Visual Basic
Public Readonly Property IndexCount As Integer
```

```
// C#
public int IndexCount {get;}
```

Property value

The number of indexes on the table or 0 if the table schema is closed.

Remarks

Index IDs range from 1 to IndexCount, inclusively.

Note: Index IDs and count may change during a schema upgrade. To correctly identify an index, access it by name or refresh the cached IDs and counts after a schema upgrade.

IsNeverSynchronized property

Checks whether the table is marked as never being synchronized.

Prototypes

```
' Visual Basic
Public Readonly Property IsNeverSynchronized As Boolean
```

```
// C#
public bool IsNeverSynchronized {get;}
```

Property value

True if the table is marked as never being synchronized, false otherwise.

Remarks

Tables marked as never being synchronized are never synchronized, even if they are included in a publication. These tables are sometimes referred to as “no sync” tables.

Exceptions

- ULException class - A SQL error occurred.

Name property

Returns the name of the table.

Prototypes

```
' Visual Basic
Public Readonly Property Name As String
```

```
// C#
public string  Name {get;}
```

Property value

The name of the table as a string.

Exceptions

- ULException class - A SQL error occurred.

PrimaryKey property

Returns the index schema of the primary key for the table.

Prototypes

' Visual Basic
Public Readonly Property **PrimaryKey** As ULIndexSchema

// C#
public ULIndexSchema **PrimaryKey** {get;}

Property value

A ULIndexSchema class object representing the primary key for the table.

Exceptions

- ULException class - A SQL error occurred.

UploadUnchangedRows property

Checks whether the database will upload rows that have not changed.

Prototypes

' Visual Basic
Public Readonly Property **UploadUnchangedRows** As Boolean

// C#
public bool **UploadUnchangedRows** {get;}

Property value

True if the table is marked to always upload all rows during synchronization, false if the table is marked to upload only changed rows.

Remarks

Tables marked as such upload unchanged rows, as well as changed rows, when the table is synchronized. These tables are sometimes referred to as “all sync” tables.

Exceptions

- ULException class - A SQL error occurred.

Finalize method

Releases unmanaged resources and performs other cleanup operations before the ULTableSchema is reclaimed by garbage collection.

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

GetColumnDefaultValue method

Returns the default value of the specified column.

Prototypes

```
' Visual Basic
Public Function GetColumnDefaultValue( _
  ByVal columnID As Integer _
) As String
```

```
// C#
public string  GetColumnDefaultValue(
  int columnID
);
```

Parameters

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

Return value

The default value of the specified column as a string or a null reference (Nothing in Visual Basic) if the default value is null.

Exceptions

- ULException class - A SQL error occurred.

GetColumnPartitionSize method

Returns the global autoincrement partition size assigned to the specified column.

Prototypes

```
' Visual Basic
Public Function GetColumnPartitionSize( _
  ByVal columnID As Integer _
) As UInt64
```

```
// C#
public ulong GetColumnPartitionSize(
  int columnID
);
```

Parameters

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

Return value

The column's global autoincrement partition size as a UInt64.

Remarks

All global autoincrement columns in a given table share the same global autoincrement partition.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTableSchema class" on page 324
- "ULTableSchema members" on page 324
- "IsColumnGlobalAutoIncrement method" on page 332

GetIndex method

Returns the index schema of the named index.

Prototypes

```
' Visual Basic
Public Function GetIndex( _
  ByVal name As String _
) As ULIndexSchema

// C#
public ULIndexSchema GetIndex(
  string name
);
```

Parameters

- **name** The name of the index.

Return value

A ULIndexSchema class object representing the named index.

Exceptions

- ULException class - A SQL error occurred.

GetIndexName method

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

Prototypes

```
' Visual Basic
Public Function GetIndexName( _
  ByVal indexID As Integer _
) As String

// C#
public string GetIndexName(
  int indexID
);
```

Parameters

- **indexID** The ID of the index. The value must be in the range [1,IndexCount property].

Return value

The name of the index as a string.

Remarks

Index IDs and counts may change during a schema upgrade. To correctly identify an index, access it by name or refresh the cached IDs and counts after a schema upgrade.

Exceptions

- ULException class - A SQL error occurred.

See also

- “ULTableSchema class” on page 324
- “ULTableSchema members” on page 324
- “IndexCount property” on page 326

GetOptimalIndex method

The optimal index for searching a table using the specified column.

Prototypes

```
' Visual Basic
Public Function GetOptimalIndex( _
  ByVal columnID As Integer _
) As ULIndexSchema

// C#
public ULIndexSchema GetOptimalIndex(
  int columnID
);
```

Parameters

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

Return value

A ULIndexSchema class object representing the optimal index for the specified column.

Remarks

The specified column is the first column in the index, but the index may have more than one column.

Exceptions

- ULException class - A SQL error occurred.

IsColumnAutoIncrement method

Checks whether the specified column's default is set to autoincrement.

Prototypes

```
' Visual Basic
Public Function IsColumnAutoIncrement( _
  ByVal columnID As Integer _
) As Boolean

// C#
public bool IsColumnAutoIncrement(
  int columnID
);
```

Parameters

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

Return value

True if the column is autoincrementing, false if it is not autoincrementing.

Exceptions

- ULException class - A SQL error occurred.

IsColumnCurrentDate method

Checks whether the specified column's default is set to the current date (ULDbType enumeration).

Prototypes

```
' Visual Basic
Public Function IsColumnCurrentDate( _
  ByVal columnID As Integer _
) As Boolean

// C#
public bool IsColumnCurrentDate(
  int columnID
);
```

Parameters

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

Return value

True if the column defaults to the current date, false if the column does not default to the current date.

Exceptions

- ULException class - A SQL error occurred.

IsColumnCurrentTime method

Checks whether the specified column's default is set to the current time (ULDbType enumeration).

Prototypes

```
' Visual Basic
Public Function IsColumnCurrentTime( _
  ByVal columnID As Integer _
) As Boolean

// C#
public bool IsColumnCurrentTime(
  int columnID
);
```

Parameters

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

Return value

True if the column defaults to the current time, false if the column does not default to the current time.

Exceptions

- ULException class - A SQL error occurred.

IsColumnCurrentTimestamp method

Checks whether the specified column's default is set to the current timestamp (ULDbType enumeration).

Prototypes

```
' Visual Basic
Public Function IsColumnCurrentTimestamp( _
  ByVal columnID As Integer _
) As Boolean
```

```
// C#
public bool IsColumnCurrentTimestamp(
  int columnID
);
```

Parameters

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

Return value

True if the column defaults to the current timestamp, false if the column does not default to the current timestamp.

Exceptions

- ULException class - A SQL error occurred.

IsColumnGlobalAutoIncrement method

Checks whether the specified column's default is set to global autoincrement.

Prototypes

```
' Visual Basic
Public Function IsColumnGlobalAutoIncrement( _
  ByVal columnID As Integer _
) As Boolean
```

```
// C#
public bool IsColumnGlobalAutoIncrement(
  int columnID
);
```

Parameters

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

Return value

True if the column is global autoincrementing, false if it is not global autoincrementing.

Exceptions

- ULException class - A SQL error occurred.

See also

- "ULTableSchema class" on page 324
- "ULTableSchema members" on page 324

- "GetColumnPartitionSize method" on page 328
- "DatabaseID property" on page 79

IsColumnNewUUID method

Checks whether the specified column's default is set to a new UUID (Guid).

Prototypes

```
' Visual Basic
Public Function IsColumnNewUUID( _
  ByVal columnID As Integer _
) As Boolean

// C#
public bool IsColumnNewUUID(
  int columnID
);
```

Parameters

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

Return value

True if the column defaults to a new UUID, false if the column does not default to a new UUID.

Exceptions

- ULException class - A SQL error occurred.

IsColumnNullable method

Checks whether the specified column is nullable.

Prototypes

```
' Visual Basic
Public Function IsColumnNullable( _
  ByVal columnID As Integer _
) As Boolean

// C#
public bool IsColumnNullable(
  int columnID
);
```

Parameters

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

Return value

True if the column is nullable, false if it is not nullable.

Exceptions

- ULException class - A SQL error occurred.

IsInPublication method

Checks whether the table is contained in the named publication.

Prototypes

```
' Visual Basic
Public Function IsInPublication( _
  ByVal pubName As String _
) As Boolean

// C#
public bool IsInPublication(
  string  pubName
);
```

Parameters

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

Return value

True if the table is in the publication, false if the table is not in the publication.

Exceptions

- ULException class - A SQL error occurred.

ULTransaction class

Represents a SQL transaction.

Prototypes

' Visual Basic
NotInheritable Public Class **ULTransaction**
  Implements IDbTransaction, IDisposable

// C#
public sealed class **ULTransaction** :
  IDbTransaction, IDisposable

Remarks

There is no constructor for ULTransaction. To obtain a ULTransaction object, use the BeginTransaction method method. To associate a command with a transaction, use the Transaction property property.

Once a transaction has been committed or rolled back, the connection reverts to automatically committing all operations as they are executed. To group more operations together, a new transaction must be created.

Implements: IDbTransaction, IDisposable

ULTransaction members

Public instance properties

Member	Description
Connection property	Returns the connection associated with the transaction.
IsolationLevel property	Returns the isolation level for the transaction.

Public instance methods

Member	Description
Commit method	Commits the database transaction.
Dispose method	Releases the resources used by the ULTransaction and rolls back any uncommitted commands.
Rollback method	Rolls back the transaction's outstanding changes to the database.

Protected instance methods

Member	Description
Finalize method	Releases unmanaged resources and performs other cleanup operations before the ULTransaction is reclaimed by garbage collection.

Connection property

Returns the connection associated with the transaction.

Prototypes

```
' Visual Basic
Public Readonly Property Connection As ULConnection

// C#
public ULConnection Connection {get;}
```

Property value

The ULConnection class object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Remarks

This is the strongly typed version of IDbTransaction.Connection.

See also

- "ULTransaction class" on page 335
- "ULTransaction members" on page 335
- "BeginTransaction method" on page 82

IsolationLevel property

Returns the isolation level for the transaction.

Prototypes

```
' Visual Basic
NotOverridable Public Readonly Property IsolationLevel As IsolationLevel _
  Implements IDbTransaction.IsolationLevel

// C#
public IsolationLevel IsolationLevel {get;}
```

Property value

One of the IsolationLevel values. UltraLite.NET only supports IsolationLevel.ReadUncommitted.

Implements

IDbTransaction.IsolationLevel

See also

- "ULTransaction class" on page 335
- "ULTransaction members" on page 335
- "BeginTransaction method" on page 82

Commit method

Commits the database transaction.

Prototypes

```
' Visual Basic
NotOverridable Public Sub Commit() _
  Implements IDbTransaction.Commit

// C#
public void Commit();
```

Remarks

Once a transaction has been committed or rolled back, the connection reverts to automatically committing all operations as they are executed. To group more operations together, a new transaction must be created.

Implements

IDbTransaction.Commit

See also

- "ULTransaction class" on page 335
- "ULTransaction members" on page 335
- "Rollback method" on page 337

Dispose method

Releases the resources used by the ULTransaction and rolls back any uncommitted commands.

Prototypes

```
' Visual Basic
Overloads NotOverridable Public Sub Dispose() _
  Implements IDisposable.Dispose

// C#
public void Dispose();
```

Implements

IDisposable.Dispose

Finalize method

Releases unmanaged resources and performs other cleanup operations before the ULTransaction is reclaimed by garbage collection.

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()

// C#
protected override void Finalize();
```

Rollback method

Rolls back the transaction's outstanding changes to the database.

Prototypes

```
' Visual Basic
NotOverridable Public Sub Rollback() _
  Implements IDbTransaction.Rollback
```

```
// C#
public void Rollback();
```

Remarks

Once a transaction has been committed or rolled back, the connection reverts to automatically committing all operations as they are executed. To group more operations together, a new transaction must be created.

Implements

IDbTransaction.Rollback

See also

- “ULTransaction class” on page 335
- “ULTransaction members” on page 335
- “Commit method” on page 336

CHAPTER 5

iAnywhere.UltraLite namespace

About this chapter

The **iAnywhere.UltraLite** namespace is deprecated in release 9.0.2, and will not be available in the next major release of SQL Anywhere. See @olink targetdoc="uldotnet" targetptr="ianywhere-data-ultralite"@iAnywhere.Data.UltraLite namespace@/olink@ for more information.

The **iAnywhere.UltraLite** namespace contains UltraLite.NET classes, interfaces, and enumerations that allow you to write C# or Visual Basic .NET code for applications that use UltraLite databases.

This API Reference for UltraLite.NET provides a complete reference of the features supported. Each topic contains information about a specific class, interface, or enumeration.

Application programs must first instantiate an instance of DatabaseManager class before using any other UltraLite.NET class.

The **iAnywhere.UltraLite** assembly uses a satellite resource assembly called **iAnywhere.UltraLite.resources**. The main assembly searches for this resource assembly by culture, using the following order: System.Globalization.CultureInfo.CurrentUICulture, then System.Globalization.CultureInfo.CurrentCulture, and finally culture "EN".

Contents

ActiveSyncListener interface

The listener interface for receiving ActiveSync events.

Prototypes

```
' Visual Basic
Public Interface ActiveSyncListener
```

```
// C#
public interface ActiveSyncListener
```

ActiveSyncListener members

Public instance methods

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

ActiveSyncInvoked method

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

Prototypes

```
' Visual Basic
Public Sub ActiveSyncInvoked( _
  ByVal launchedByProvider As Boolean _
)
```

```
// C#
public void ActiveSyncInvoked(
  bool launchedByProvider
);
```

Parameters

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

Remarks

This method is invoked by a separate thread. To avoid multi-threading issues, it should post an event to the UI. If you are using multi-threading, it is recommended that you use a separate connection and use the `lock` keyword to access any objects shared with the rest of the application. The synchronization resulting from a call to `ActiveSyncInvoked` must specify a StreamType.ACTIVE_SYNC for its connection's SyncInfo stream.

Example

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

```
[Visual Basic]
Imports iAnywhere.UltraLite

              Public Class MainWindow
  Inherits System.Windows.Forms.Form
  Implements ActiveSyncListener
  Private dbMgr As DatabaseManager
  Private conn As Connection

            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
    dbMgr = New DatabaseManager
    dbMgr.SetActiveSyncListener( "myCompany.myapp",
Me )
    'Create Connection
    ...
  End Sub

            Protected Overrides Sub
OnClosing(ByVal e As
System.ComponentModel.CancelEventArgs)
    If Not (dbMgr Is Nothing) Then
      dbMgr.SetActiveSyncListener(Nothing, Nothing)
      dbMgr = Nothing
    End If
  End Sub

            Public Sub ActiveSyncInvoked(ByVal
launchedByProvider As Boolean) Implements
iAnywhere.UltraLite.ActiveSyncListener.ActiveSyncInvoke
d
    Me.Invoke(New EventHandler(AddressOf
Me.ActiveSyncAction))
  End Sub

            Public Sub ActiveSyncAction(ByVal
sender As Object, ByVal e As EventArgs)
    ' Do active sync
    conn.SyncParms.Stream = StreamType.ACTIVE_SYNC
    conn.Synchronize()
  End Sub
End Class

          [C#]
using iAnywhere.UltraLite;
public class Form1 : System.Windows.Forms.Form,
ActiveSyncListener
{
  private System.Windows.Forms.MainMenu mainMenu1;
  private DatabaseManager dbMgr;
  private Connection conn;

          public Form1()
```

AuthStatusCode enumeration

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

Prototypes

' Visual Basic
Public Enum **AuthStatusCode**

// C#
public enum **AuthStatusCode**

Members

Member	Description
EXPIRED	User ID or password has expired - authorization failed (EXPIRED = 3000).
IN_USE	User ID is already in use - authorization failed (IN_USE = 5000).
INVALID	Bad user ID or password - authorization failed (INVALID = 4000).
UNKNOWN	Authorization status is unknown, possibly because the connection has not yet performed a synchronization (UNKNOWN = 0).
VALID	User ID and password were valid at time of synchronization (VALID = 1000).
VALID_BUT_EXPIRES_SOON	User ID and password were valid at time of synchronization but will expire soon (VALID_BUT_EXPIRES_SOON = 2000).

See also

- "AuthStatus property" on page 543

Connection class

Represents a connection to an UltraLite database.

Prototypes

' Visual Basic
NotInheritable Public Class **Connection**

// C#
public sealed class **Connection**

Remarks

This class cannot be directly instantiated. Connections are instantiated and opened using OpenConnection method or if the database does not yet exist, CreateDatabase 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. You must close all tables opened on a connection before closing the connection.

When a SQLException is thrown, the error code in the exception supercedes the error code in LastSQLCode property or the values of IsLastFetchOK property and IsLastCodeOK property.

Connection members

Public static fields (Shared)

Member	Description
AutoCommit field	Controls whether a commit is performed after each insert, update or delete statement.
INVALID_DATABASE_ID field	Database ID constant indicating that the DatabaseID property has not been set.

Public instance fields

Member	Description
AutoCommit field	Controls whether a commit is performed after each insert, update or delete statement.

Public instance properties

Member	Description
ConnectionOpenParms property	The parameters used to open this connection.

Member	Description
DatabaseID property	Database ID value to be used for global autoincrement columns.
GlobalAutoIncrementUsage property	The percentage of available global autoincrement values that have been used.
IsDatabaseNew property	Checks whether the database was new when this connection was opened.
IsLastCodeOK property	Classifies the last SQL code as successful or unsuccessful.
IsLastFetchOK property	Provides a convenient way of checking that the most recent fetch of a row succeeded or failed.
IsOpen property	Determines whether the connection is open or closed.
LastIdentity property	The most recent identity value used.
LastSQLCode property	The SQL code value for the success or failure of the last database operation.
Schema property	The database schema associated with the connection.
SyncParms property	The synchronization settings for this connection.
SyncResult property	The results of the last synchronization for this connection.

Public instance methods

Member	Description
ChangeEncryptionKey method	Changes the database's encryption key to the specified new key.
Close method	Closes this connection.
Commit method	Commits outstanding changes to the database.
CountUploadRows method	Returns the number of rows that need to be uploaded when the next synchronization takes place.
CountUploadRows method	Returns the number of rows that need to be uploaded when the next synchronization takes place.
GetLastDownloadTime method	Returns the time of the most recent download of the specified publication.
GetNewUUID method	Generates a new UUID (Guid).
GetTable method	Creates and returns a reference to the requested table in the database.
GrantConnectTo method	Grants access to an UltraLite database for a user ID with a specified password.

Member	Description
PrepareStatement method	Pre-compiles and stores into a PreparedStatement object a SQL statement with or without IN parameters.
ResetLastDownloadTime method	Resets the time of the most recent download.
RevokeConnectFrom method	Revokes access to an UltraLite database from the specified user ID.
Rollback method	Rolls back outstanding changes to the database.
RollbackPartialDownload method	Rolls back outstanding changes to the database from a partial download.
StartSynchronizationDelete method	Marks all subsequent deletes made by this connection for synchronization.
StopSynchronizationDelete method	Prevents delete operations from being synchronized.
Synchronize method	Synchronize the database using the current SyncParms property.
Synchronize method	Synchronize the database using the current SyncParms property with progress events posted to the specified listener.

Protected instance methods

Member	Description
Finalize method	Cleans up the associated native object.

AutoCommit field

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

Prototypes

' Visual Basic
Public**AutoCommit** As Boolean

// C#
public bool **AutoCommit**;

Remarks

If `AutoCommit` is false, a commit or rollback is performed only when the user invokes the Commit method or Rollback method methods.

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

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “Commit method” on page 353
- “Rollback method” on page 358

INVALID_DATABASE_ID field

Database ID constant indicating that the DatabaseID property has not been set.

Prototypes

’ Visual Basic
Public Shared **INVALID_DATABASE_ID** As Long

// C#
public const long **INVALID_DATABASE_ID**;

ConnectionOpenParms property

The parameters used to open this connection.

Prototypes

’ Visual Basic
Public Readonly Property **ConnectionOpenParms** As ConnectionParms

// C#
public ConnectionParms **ConnectionOpenParms** {get;}

Property value

The parameters used to open this connection in a read-only ConnectionParms class instance.

Remarks

This property can be used after the connection is closed.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “ConnectionParms class” on page 362
- “CreateDatabase method” on page 401
- “OpenConnection method” on page 403

DatabaseID property

Database ID value to be used for global autoincrement columns.

Prototypes

’ Visual Basic
Public Property **DatabaseID** As Long

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

Property value

Database ID value must be in range `[0,UInt32.MaxValue]`. A value of INVALID_DATABASE_ID field is used to indicate that the database ID has

not been set.

Exceptions

- SQLException class - If the new ID is invalid.

GlobalAutoIncrementUsage property

The percentage of available global autoincrement values that have been used.

Prototypes

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

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

Property value

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

Remarks

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

Exceptions

- SQLException class - A SQL error occurred.

IsDatabaseNew property

Checks whether the database was new when this connection was opened.

Prototypes

' Visual Basic
Public Readonly Property **IsDatabaseNew** As Boolean

// C#
public bool **IsDatabaseNew** {get;}

Property value

True if the database was new when this connection was opened, false if the database already existed when this connection was opened.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “CreateDatabase method” on page 401
- “OpenConnection method” on page 403

IsLastCodeOK property

Classifies the last SQL code as successful or unsuccessful.

Prototypes

' Visual Basic
Public Readonly Property **IsLastCodeOK** As Boolean

// C#
public bool **IsLastCodeOK** {get;}

Property value — True if the most recent SQL code represents a warning or success, and false if the most recent SQL code represents an error. For example, SQLCode enumeration is not an error.

See also
- "Connection class" on page 345
- "Connection members" on page 345
- "LastSQLCode property" on page 351
- "SQLCode enumeration" on page 468

IsLastFetchOK property

Provides a convenient way of checking that the most recent fetch of a row succeeded or failed.

Prototypes

```
' Visual Basic
Public Readonly Property IsLastFetchOK As Boolean
```

```
// C#
public bool IsLastFetchOK {get;}
```

Property value — True if the last fetch succeeded, false otherwise.

Remarks — When a SQLException is thrown, the error code in the exception supercedes the error code in LastSQLCode property or the values of IsLastFetchOK property and IsLastCodeOK property.

See also
- "Connection class" on page 345
- "Connection members" on page 345
- "LastSQLCode property" on page 351

IsOpen property

Determines whether the connection is open or closed.

Prototypes

```
' Visual Basic
Public Readonly Property IsOpen As Boolean
```

```
// C#
public bool IsOpen {get;}
```

Property value — True if the connection is open, false otherwise.

LastIdentity property

The most recent identity value used.

Prototypes

```
' Visual Basic
Public Readonly Property LastIdentity As UInt64
```

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

Property value

The most recent identity value used. This property is equivalent to the SQL statement:

```
SELECT @@identity
```

LastIdentity is particularly useful in the context of global autoincrement columns.

Remarks

Since this property only allows you to determine the most recently assigned default value, you should retrieve this value soon after executing the insert statement to avoid spurious results.

Occasionally, a single insert statement may include more than one column of type global autoincrement. In this case, `LastIdentity` is one of the generated default values, but there is no reliable means to determine which one. For this reason, you should design your database and write your insert statements so as to avoid this situation.

Exceptions

- SQLException class - A SQL error occurred.

LastSQLCode property

The SQL code value for the success or failure of the last database operation.

Prototypes

' Visual Basic
Public Readonly Property **LastSQLCode** As SQLCode

// C#
public SQLCode **LastSQLCode** {get;}

Remarks

The SQL code is a standard Adaptive Server Anywhere code and is reset by any subsequent UltraLite database operation on this connection.

When a SQLException is thrown, the error code in the exception supercedes the error code returned by LastSQLCode property or the results from IsLastFetchOK property and IsLastCodeOK property.

Schema property

The database schema associated with the connection.

Prototypes

' Visual Basic
Public Readonly Property **Schema** As DatabaseSchema

// C#
public DatabaseSchema **Schema** {get;}

Property value — A reference to the schema of the database referenced by the connection.

Remarks — This property is only valid while its connection is open.

See also
- “Connection class” on page 345
- “Connection members” on page 345
- “DatabaseSchema class” on page 413

SyncParms property

The synchronization settings for this connection.

Prototypes

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

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

Property value — A reference to the parameters used for synchronization by the connection. Modifications to the parameters will affect the next synchronization made over the connection.

See also
- “Connection class” on page 345
- “Connection members” on page 345
- “SyncParms property” on page 352
- “SyncResult property” on page 352

SyncResult property

The results of the last synchronization for this connection.

Prototypes

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

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

Property value — A reference to the results of the last synchronization for this connection.

See also
- “Connection class” on page 345
- “Connection members” on page 345
- “SyncResult property” on page 352
- “SyncParms property” on page 352

ChangeEncryptionKey method

Changes the database’s encryption key to the specified new key.

Prototypes

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

// C#
public void ChangeEncryptionKey(
  string  newKey
);
```

Parameters

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

Remarks

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

Exceptions

- SQLException class - A SQL error occurred.

Close method

Closes this connection.

Prototypes

```
' Visual Basic
Public Sub Close()

// C#
public void Close();
```

Remarks

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

It is an error to use any object (table, schema, etc.) associated with a closed connection.

Exceptions

- SQLException class - A SQL error occurred.

Commit method

Commits outstanding changes to the database.

Prototypes

```
' Visual Basic
Public Sub Commit()

// C#
public void Commit();
```

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “Commit method” on page 353
- “Rollback method” on page 358

CountUploadRows method

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

Prototypes

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

```
// C#
public uint CountUploadRows(
  int mask,
  uint threshold
);
```

Parameters

- **mask** The set of publications to check for rows. See the PublicationSchema class class for more information.
- **threshold** The maximum number of rows to count, limiting the amount of time taken by `CountUploadRows`. A value of 0 corresponds to the maximum limit. A value of 1 determines if any rows need to be synchronized.

Return value

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

Exceptions

- SQLException class - A SQL error occurred.

CountUploadRows method

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

Prototypes

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

```
// C#
public long CountUploadRows(
  int mask,
  long threshold
);
```

Parameters

- **mask** The set of publications to check for rows. See the PublicationSchema class class for more information.

- **threshold** The maximum number of rows to count, and so limits 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. *threshold* must be in range `[0,0x0fffffff]`.

Return value

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

Remarks

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

Exceptions

- SQLException class - A SQL error occurred.

Finalize method

Cleans up the associated native object.

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()
```

```
// C#
protected override void Finalize();
```

GetLastDownloadTime method

Returns the time of the most recent download of the specified publication.

Prototypes

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

```
// C#
public DateTime GetLastDownloadTime(
  int mask
);
```

Parameters

- **mask** The mask of the publication to check. See the PublicationSchema class class for more information.

Return value

The timestamp of the last download.

Remarks

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

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Connection class” on page 345

- "Connection members" on page 345
- "ResetLastDownloadTime method" on page 358

GetNewUUID method

Generates a new UUID (Guid).

Prototypes

```
' Visual Basic
Public Function GetNewUUID() As Guid

// C#
public Guid GetNewUUID();
```

Return value

A new UUID as a Guid.

Remarks

Provided here as it is not included in the .NET Compact Framework.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Connection class" on page 345
- "Connection members" on page 345
- "SetUUID method" on page 582
- "GetUUID method" on page 388

GetTable method

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

Prototypes

```
' Visual Basic
Public Function GetTable( _
  ByVal name As String _
) As Table

// C#
public Table GetTable(
  string  name
);
```

Parameters

- **name** The name of the table to fetch.

Return value

An instance of the Table class object representing the desired table.

Exceptions

- SQLException class - A SQL error occurred.

GrantConnectTo method

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

Prototypes

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

// C#
public void GrantConnectTo(
  string uid,
  string pwd
);
```

Parameters

- **uid** The user ID to receive access to the database. The maximum length of the ID is 16 characters.
- **pwd** The password to be associated with the ID. The maximum length is 16 characters.

Remarks

If an existing user ID is specified, this function updates the password for the user. UltraLite supports a maximum of 4 users. This method is enabled only if user authentication was enabled when the connection was opened.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “OpenConnection method” on page 403

PrepareStatement method

Pre-compiles and stores into a PreparedStatement object a SQL statement with or without IN parameters.

Prototypes

```
' Visual Basic
Public Function PrepareStatement( _
  ByVal sql As String _
) As PreparedStatement

// C#
public PreparedStatement PrepareStatement(
  string sql
);
```

Parameters

- **sql** a SQL statement that may contain one or more ‘?’ IN parameter placeholders.

Return value

a new PreparedStatement object containing the pre-compiled statement.

Remarks

The prepared statement object can be used to efficiently execute this statement multiple times.

Exceptions

- SQLException class - A SQL error occurred.

ResetLastDownloadTime method

Resets the time of the most recent download.

Prototypes

```
' Visual Basic
Public Sub ResetLastDownloadTime( _
  ByVal mask As Integer _
)

// C#
public void ResetLastDownloadTime(
  int mask
);
```

Parameters

- **mask** The set of publications to reset. See PublicationSchema class class for more information.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “GetLastDownloadTime method” on page 355

RevokeConnectFrom method

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

Prototypes

```
' Visual Basic
Public Sub RevokeConnectFrom( _
  ByVal uid As String _
)

// C#
public void RevokeConnectFrom(
  string uid
);
```

Parameters

- **uid** The user ID for which database access is to be revoked. The maximum length of a user ID is 16 characters.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “GrantConnectTo method” on page 356

Rollback method

Rolls back outstanding changes to the database.

Prototypes

```
' Visual Basic
Public Sub Rollback()
```

```
// C#
public void Rollback();
```

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “Commit method” on page 353
- “AutoCommit field” on page 347

RollbackPartialDownload method

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

Prototypes

```
' Visual Basic
Public Sub RollbackPartialDownload()
```

```
// C#
public void RollbackPartialDownload();
```

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “KeepPartialDownload property” on page 494
- “ResumePartialDownload property” on page 497

StartSynchronizationDelete method

Marks all subsequent deletes made by this connection for synchronization.

Prototypes

```
' Visual Basic
Public Sub StartSynchronizationDelete()
```

```
// C#
public void StartSynchronizationDelete();
```

Remarks

Once this function is called, all delete operations are again synchronized, causing the deleted rows to also be removed from the consolidated database.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “StopSynchronizationDelete method” on page 360

StopSynchronizationDelete method

Prevents delete operations from being synchronized.

Prototypes

```
' Visual Basic
Public Sub StopSynchronizationDelete()
```

```
// C#
public void StopSynchronizationDelete();
```

Remarks

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

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “StartSynchronizationDelete method” on page 359

Synchronize method

Synchronize the database using the current SyncParms property.

Prototypes

```
' Visual Basic
Overloads Public Sub Synchronize()
```

```
// C#
public void Synchronize();
```

Remarks

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

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Connection class” on page 345
- “Connection members” on page 345
- “SyncParms property” on page 352
- “SyncResult property” on page 352

Synchronize method

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

Prototypes

```
' Visual Basic
Overloads Public Sub Synchronize( _
  ByVal listener As SyncProgressListener _
)
```

```
// C#
public void Synchronize(
  SyncProgressListener listener
);
```

Parameters

- **listener** The object that will receive synchronization progress events.

Remarks

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

Errors during synchronization will be posted as SyncProgressState enumeration events and then thrown as SQLException classs.

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

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Connection class" on page 345
- "Connection members" on page 345
- "SyncParms property" on page 352
- "SyncResult property" on page 352
- "SyncProgressListener interface" on page 539

ConnectionParms class

Specifies parameters for opening a connection to an UltraLite database.

Prototypes

```
' Visual Basic
Public Class ConnectionParms
  Inherits DatabaseNameParms

// C#
public class ConnectionParms :
  DatabaseNameParms
```

Remarks

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 RevokeConnectFrom method. To add a user or change a user's password, use GrantConnectTo method.

If multiple connections are created, each one must be given a unique name using `ConnectionName="name";`.

See also

- "ConnectionParms members" on page 362
- "AdditionalParms property" on page 407

ConnectionParms members

Public instance constructors

Member	Description
ConnectionParms constructor	Initializes a ConnectionParms instance with its default values.

Public instance properties

Member	Description
AdditionalParms property (inherited from DatabaseNameParms)	Specifies additional parameters as a semicolon-separated list of *name=value* pairs. These are less commonly used parameters.
CacheSize property	Specifies the size of the cache.
ConnectionName property	Specifies a name for the connection. This is only needed if you create more than one connection to the database.
DatabaseOnCE property (inherited from DatabaseNameParms)	Specifies the path and filename of the UltraLite database on Windows CE.

Member	Description
DatabaseOnDesktop property (inherited from DatabaseNameParms)	Specifies the path and filename of the UltraLite database on Windows desktop platforms.
EncryptionKey property	Specifies a key for encrypting the database.
ParmsUsed property (inherited from DatabaseNameParms)	Returns the parameters actually used by the last `DatabaseManager` method to use this instance.
Password property	Specifies the password for the authenticated user.
UserID property	Specifies an authenticated user for the database.

Public instance methods

Member	Description
ToString method (inherited from DatabaseNameParms)	Returns the string representation of this instance.

Public instance events

Member	Description
UnusedEvent event (inherited from DatabaseNameParms)	Unused.

Protected instance methods

Member	Description
Dispose method (inherited from DatabaseNameParms)	Releases the resources used by this object.

ConnectionParms constructor

Initializes a ConnectionParms instance with its default values.

Prototypes

' Visual Basic
Overloads Public Sub **New()**

// C#
public **ConnectionParms();**

Exceptions

- SQLException class - A SQL error occurred.

CacheSize property

Specifies the size of the cache.

Prototypes

```
' Visual Basic
Public Property CacheSize As String

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

Remarks

The values for the cache size are specified in units of bytes. Use the suffix `k` or `K` to indicate units of kilobytes and use the suffix 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 sixteen 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. Also, large cache sizes may 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. This is only needed if you create more than one connection to the database.

Prototypes

```
' Visual Basic
Public Property ConnectionName As String

// C#
public string  ConnectionName {get;set;}
```

EncryptionKey property

Specifies a key for encrypting the database.

Prototypes

' Visual Basic
Public Property **EncryptionKey** As String

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

Remarks

All `openConnection` calls 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. As well, including a combination of numbers, letters, and special characters decreases the chances of someone guessing the key.

See also

- "ConnectionParms class" on page 362
- "ConnectionParms members" on page 362
- "ChangeEncryptionKey method" on page 352

Password property

Specifies the password for the authenticated user.

Prototypes

' Visual Basic
Public Property **Password** As String

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

Remarks

When a database is created, the password for the `DBA` user ID is set to `SQL`. Passwords are case-insensitive if the database is case-insensitive and case-sensitive if the database is case-sensitive. The default value is `SQL`.

See also

- "ConnectionParms class" on page 362
- "ConnectionParms members" on page 362
- "UserID property" on page 365

UserID property

Specifies an authenticated user for the database.

Prototypes

' Visual Basic
Public Property **UserID** As String

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

Remarks

Databases are initially created with a single authenticated user named `DBA`.

User IDs are case-insensitive if the database is case-insensitive and case-sensitive if the database is case-sensitive. The default value is `DBA`.

See also

- “ConnectionParms class” on page 362
- “ConnectionParms members” on page 362
- “Password property” on page 365
- “GrantConnectTo method” on page 356
- “RevokeConnectFrom method” on page 358
- “IsCaseSensitive property” on page 414

CreateParms class

Specifies parameters for creating an UltraLite database and opening a connection to it.

Prototypes

' Visual Basic
```
NotInheritable Public Class CreateParms
  Inherits ConnectionParms
```

// C#
```
public sealed class CreateParms :
  ConnectionParms
```

Remarks

NOTE: Only one database may be active at a given time. Attempts to create a different database while other connections are open will result in an error.

A valid schema file must be specified to successfully create a database.

See also

- "CreateParms members" on page 367
- "AdditionalParms property" on page 407

CreateParms members

Public instance constructors

Member	Description
CreateParms constructor	Initializes a CreateParms instance with its default values.

Public instance properties

Member	Description
AdditionalParms property (inherited from DatabaseNameParms)	Specifies additional parameters as a semicolon-separated list of *name=value* pairs. These are less commonly used parameters.
CacheSize property (inherited from ConnectionParms)	Specifies the size of the cache.
ConnectionName property (inherited from ConnectionParms)	Specifies a name for the connection. This is only needed if you create more than one connection to the database.
DatabaseOnCE property (inherited from DatabaseNameParms)	Specifies the path and filename of the UltraLite database on Windows CE.
DatabaseOnDesktop property (inherited from DatabaseNameParms)	Specifies the path and filename of the UltraLite database on Windows desktop platforms.

Member	Description
EncryptionKey property (inherited from ConnectionParms)	Specifies a key for encrypting the database.
ParmsUsed property (inherited from DatabaseNameParms)	Returns the parameters actually used by the last `DatabaseManager` method to use this instance.
Password property (inherited from ConnectionParms)	Specifies the password for the authenticated user.
Schema property	Specifies the schema for an UltraLite database.
UserID property (inherited from ConnectionParms)	Specifies an authenticated user for the database.

Public instance methods

Member	Description
ToString method (inherited from DatabaseNameParms)	Returns the string representation of this instance.

Public instance events

Member	Description
UnusedEvent event (inherited from DatabaseNameParms)	Unused.

Protected instance methods

Member	Description
Dispose method (inherited from DatabaseNameParms)	Releases the resources used by this object.

CreateParms constructor

Initializes a CreateParms instance with its default values.

Prototypes

```
' Visual Basic
Public Sub New()
```

// C#
public **CreateParms();**

Exceptions

- SQLException class - A SQL error occurred.

Schema property

Specifies the schema for an UltraLite database.

Prototypes

' Visual Basic
Public Property **Schema** As SchemaParms

// C#
public SchemaParms **Schema** {get;set;}

See also

- "CreateParms class" on page 367
- "CreateParms members" on page 367
- "SchemaParms class" on page 458

Cursor class

Represents a cursor in an UltraLite database. Cursors are sets of rows from either a table or a result set.

Prototypes

' Visual Basic
MustInherit Public Class **Cursor**

// C#
public abstract class **Cursor**

Remarks

This class is an abstract base class of the Table class and ResultSet class classes.

Derived classes have a `Schema` property through which the description of the rows may be accessed.

Cursor members

Public instance properties

Member	Description
IsBOF property	Returns true if the current row position is before the first row.
IsEOF property	Returns true if the current row position is after the last row.
IsOpen property	Checks whether this cursor is currently open.
RowCount property	Returns the number of rows in the cursor.

Public instance methods

Member	Description
Close method	Closes the cursor.
Fill method	Fills a ADO .NET DataTable object with the rows in the cursor.
GetBoolean method	Returns the value for the specified column as a Boolean.
GetBytes method	Returns the value for the specified column as an array of Bytes. Only valid for columns of type SQLType enumeration, SQLType enumeration, or SQLType enumeration.
GetBytes method	Copies a subset of the value for the specified SQLType enumeration column, beginning at the specified offset, to the specified offset of the destination Byte array.

Member	Description
GetChars method	Copies a subset of the value for the specified SQLType enumeration column, beginning at the specified offset, to the specified offset of the destination System.Char array.
GetDouble method	Returns the value for the specified column as a Double.
GetFloat method	Returns the value for the specified column as a Single.
GetInt method	Returns the value for the specified column as an Int32.
GetLong method	Returns the value for the specified column as an Int64.
GetShort method	Returns the value for the specified column as an Int16.
GetString method	Returns the value for the specified column as a String.
GetTime method	Returns the value for the specified column as a TimeSpan with millisecond accuracy.
GetTimestamp method	Returns the value for the specified column as a DateTime with millisecond accuracy.
GetUInt method	Returns the value for the specified column as a UInt32.
GetULong method	Returns the value for the specified column as a UInt64.
GetUShort method	Returns the value for the specified column as a UInt16.
GetUUID method	Returns the value for the specified column as a UUID (Guid).
IsNull method	Checks whether the value from the specified column is NULL.
MoveAfterLast method	Position to after the last row of the cursor.
MoveBeforeFirst method	Position to before the first row of the cursor.
MoveFirst method	Position to the first row of the cursor.
MoveLast method	Position to the last row of the cursor.
MoveNext method	Position to the next row or after last.
MovePrevious method	Position to the previous row or before first.
MoveRelative method	Position relative to the current row.

Protected instance methods

Member	Description
Finalize method	Finalize method

IsBOF property

Returns true if the current row position is before the first row.

Prototypes

' Visual Basic
Public Readonly Property **IsBOF** As Boolean

// C#
public bool **IsBOF** {get;}

Exceptions

- SQLException class - A SQL error occurred.

IsEOF property

Returns true if the current row position is after the last row.

Prototypes

' Visual Basic
Public Readonly Property **IsEOF** As Boolean

// C#
public bool **IsEOF** {get;}

Exceptions

- SQLException class - A SQL error occurred.

IsOpen property

Checks whether this cursor is currently open.

Prototypes

' Visual Basic
Public Readonly Property **IsOpen** As Boolean

// C#
public bool **IsOpen** {get;}

Return value

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

RowCount property

Returns the number of rows in the cursor.

Prototypes

' Visual Basic
Public Readonly Property **RowCount** As Integer

// C#
public int **RowCount** {get;}

Remarks

One use for `RowCount` is to decide when to delete old rows to save space. Old rows can be deleted from the UltraLite database without being deleted from the consolidated database using the StopSynchronizationDelete method method.

Exceptions
- SQLException class - A SQL error occurred.

See also
- "Cursor class" on page 370
- "Cursor members" on page 370
- "StartSynchronizationDelete method" on page 359
- "StopSynchronizationDelete method" on page 360

Close method

Closes the cursor.

Prototypes

```
' Visual Basic
Public Sub Close()

// C#
public void Close();
```

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

Exceptions
- SQLException class - A SQL error occurred.

Fill method

Fills a ADO .NET DataTable object with the rows in the cursor.

Prototypes

```
' Visual Basic
Public Function Fill( _
  ByVal dt As DataTable _
) As Integer

// C#
public int Fill(
  DataTable dt
);
```

Parameters
- **dt** the ADO .NET DataTable object to fill.

Return value
the number of rows that were filled.

Finalize method

Finalize method

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()

// C#
protected override void Finalize();
```

GetBoolean method

Returns the value for the specified column as a Boolean.

Prototypes

```
' Visual Basic
Public Function GetBoolean( _
  ByVal columnID As Short _
) As Boolean
```

```
// C#
public bool GetBoolean(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

column value as a Boolean.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Cursor class” on page 370
- “Cursor members” on page 370
- “GetColumnID method” on page 395
- “IsNull method” on page 389
- “GetBytes method” on page 374
- “GetBytes method” on page 375
- “GetChars method” on page 377
- “GetDouble method” on page 378
- “GetFloat method” on page 379
- “GetInt method” on page 380
- “GetLong method” on page 381
- “GetShort method” on page 382
- “GetString method” on page 383
- “GetTime method” on page 384
- “GetTimestamp method” on page 385
- “GetUInt method” on page 386
- “GetULong method” on page 386
- “GetUShort method” on page 387
- “GetUUID method” on page 388
- “GetColumnSQLType method” on page 397

GetBytes method

Returns the value for the specified column as an array of Bytes. Only valid for columns of type SQLType enumeration, SQLType enumeration, or SQLType enumeration.

Prototypes

```
' Visual Basic
Overloads Public Function GetBytes( _
  ByVal columnID As Short _
) As Byte()
```

```
// C#
public byte[] GetBytes(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

column value as an array of Bytes.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Cursor class” on page 370
- “Cursor members” on page 370
- “GetColumnID method” on page 395
- “IsNull method” on page 389
- “GetBoolean method” on page 373
- “GetBytes method” on page 375
- “GetChars method” on page 377
- “GetDouble method” on page 378
- “GetFloat method” on page 379
- “GetInt method” on page 380
- “GetLong method” on page 381
- “GetShort method” on page 382
- “GetString method” on page 383
- “GetTime method” on page 384
- “GetTimestamp method” on page 385
- “GetUInt method” on page 386
- “GetULong method” on page 386
- “GetUShort method” on page 387
- “GetUUID method” on page 388
- “GetColumnSQLType method” on page 397

GetBytes method

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

Prototypes

```
' Visual Basic
Overloads Public Function GetBytes( _
  ByVal columnID As Short, _
  ByVal srcOffset As Integer, _
  ByVal dst As Byte(), _
  ByVal dstOffset As Integer, _
  ByVal count As Integer _
) As Integer
```

```
// C#
public int GetBytes(
  short columnID,
  int srcOffset,
  byte[] dst,
  int dstOffset,
  int count
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.
- **srcOffset** start position in the column value. Zero is the beginning of the value.
- **dst** the destination array.
- **dstOffset** start position in the destination array.
- **count** the number of bytes to be copied.

Return value

the actual number of bytes copied.

Remarks

The bytes at position *srcOffset* through `srcOffset+count-1` of the value are copied into positions *dstOffset* through `dstOffset+count-1`, respectively, of the destination array. If the end of the value is encountered before *count* bytes are copied, the remainder of the destination array is left unchanged.

If any of the following are true, a SQLException class with code SQLCode enumeration is thrown and the destination is not modified:

For other errors, a SQLException class with the appropriate error code is thrown.

- *dst* is null.
- *srcOffset* is negative.
- *dstOffset* is negative.
- *count* is negative.
- `dstOffset+count` is greater than `dst.Length`.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Cursor class” on page 370
- “Cursor members” on page 370
- “GetColumnID method” on page 395
- “IsNull method” on page 389
- “GetBoolean method” on page 373
- “GetBytes method” on page 374

GetChars method

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

Prototypes

```
' Visual Basic
Public Function GetChars( _
  ByVal columnID As Short, _
  ByVal srcOffset As Integer, _
  ByVal dst As Char(), _
  ByVal dstOffset As Integer, _
  ByVal count As Integer _
) As Integer
```

```
// C#
public int GetChars(
  short columnID,
  int srcOffset,
  char[] dst,
  int dstOffset,
  int count
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.
- **srcOffset** start position in the column value. Zero is the beginning of the value.
- **dst** the destination array.
- **dstOffset** start position in the destination array.

- **count** the number of characters to be copied.

Return value

the actual number of characters copied.

Remarks

The characters at position *srcOffset* through `srcOffset+count-1` of the value are copied into positions *dstOffset* through `dstOffset+count-1`, respectively, of the destination array. If the end of the value is encountered before *count* characters are copied, the remainder of the destination array is left unchanged.

If any of the following is true, a SQLException class with code SQLCode enumeration is thrown and the destination is not modified:

For other errors, a SQLException class with the appropriate error code is thrown.

- *dst* is null.
- *srcOffset* is negative.
- *dstOffset* is negative.
- *count* is negative.
- `dstOffset+count` is greater than `dst.Length`.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Cursor class” on page 370
- “Cursor members” on page 370
- “GetColumnID method” on page 395
- “IsNull method” on page 389
- “GetBoolean method” on page 373
- “GetBytes method” on page 374
- “GetBytes method” on page 375
- “GetDouble method” on page 378
- “GetFloat method” on page 379
- “GetInt method” on page 380
- “GetLong method” on page 381
- “GetShort method” on page 382
- “GetString method” on page 383
- “GetTime method” on page 384
- “GetTimestamp method” on page 385
- “GetUInt method” on page 386
- “GetULong method” on page 386
- “GetUShort method” on page 387
- “GetUUID method” on page 388
- “GetColumnSQLType method” on page 397

GetDouble method

Returns the value for the specified column as a Double.

Prototypes

```
' Visual Basic
Public Function GetDouble( _
  ByVal columnID As Short _
) As Double
```

```
// C#
public double GetDouble(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

column value as a Double.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Cursor class" on page 370
- "Cursor members" on page 370
- "GetColumnID method" on page 395
- "IsNull method" on page 389
- "GetBytes method" on page 374
- "GetBytes method" on page 375
- "GetChars method" on page 377
- "GetBoolean method" on page 373
- "GetFloat method" on page 379
- "GetInt method" on page 380
- "GetLong method" on page 381
- "GetShort method" on page 382
- "GetString method" on page 383
- "GetTime method" on page 384
- "GetTimestamp method" on page 385
- "GetUInt method" on page 386
- "GetULong method" on page 386
- "GetUShort method" on page 387
- "GetUUID method" on page 388
- "GetColumnSQLType method" on page 397

GetFloat method

Returns the value for the specified column as a Single.

Prototypes

```
' Visual Basic
Public Function GetFloat( _
  ByVal columnID As Short _
) As Single
```

```
// C#
public float  GetFloat(
   short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

column value as a Single.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Cursor class” on page 370
- “Cursor members” on page 370
- “GetColumnID method” on page 395
- “IsNull method” on page 389
- “GetBytes method” on page 374
- “GetBytes method” on page 375
- “GetChars method” on page 377
- “GetBoolean method” on page 373
- “GetDouble method” on page 378
- “GetInt method” on page 380
- “GetLong method” on page 381
- “GetShort method” on page 382
- “GetString method” on page 383
- “GetTime method” on page 384
- “GetTimestamp method” on page 385
- “GetUInt method” on page 386
- “GetULong method” on page 386
- “GetUShort method” on page 387
- “GetUUID method” on page 388
- “GetColumnSQLType method” on page 397

GetInt method

Returns the value for the specified column as an Int32.

Prototypes

```
' Visual Basic
Public Function GetInt( _
   ByVal columnID As Short _
) As Integer
```

```
// C#
public int GetInt(
   short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value: column value as an Int32.

Exceptions:
- SQLException class - A SQL error occurred.

See also:
- "Cursor class" on page 370
- "Cursor members" on page 370
- "GetColumnID method" on page 395
- "IsNull method" on page 389
- "GetBytes method" on page 374
- "GetBytes method" on page 375
- "GetChars method" on page 377
- "GetBoolean method" on page 373
- "GetDouble method" on page 378
- "GetFloat method" on page 379
- "GetLong method" on page 381
- "GetShort method" on page 382
- "GetString method" on page 383
- "GetTime method" on page 384
- "GetTimestamp method" on page 385
- "GetUInt method" on page 386
- "GetULong method" on page 386
- "GetUShort method" on page 387
- "GetUUID method" on page 388
- "GetColumnSQLType method" on page 397

GetLong method

Returns the value for the specified column as an Int64.

Prototypes:

```
' Visual Basic
Public Function GetLong( _
  ByVal columnID As Short _
) As Long
```

```
// C#
public long GetLong(
  short columnID
);
```

Parameters:
- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value: column value as an Int64

Exceptions:
- SQLException class - A SQL error occurred.

See also:
- "Cursor class" on page 370
- "Cursor members" on page 370

- “GetColumnID method” on page 395
- “IsNull method” on page 389
- “GetBytes method” on page 374
- “GetBytes method” on page 375
- “GetChars method” on page 377
- “GetBoolean method” on page 373
- “GetDouble method” on page 378
- “GetFloat method” on page 379
- “GetInt method” on page 380
- “GetShort method” on page 382
- “GetString method” on page 383
- “GetTime method” on page 384
- “GetTimestamp method” on page 385
- “GetUInt method” on page 386
- “GetULong method” on page 386
- “GetUShort method” on page 387
- “GetUUID method” on page 388
- “GetColumnSQLType method” on page 397

GetShort method

Returns the value for the specified column as an Int16.

Prototypes

```
' Visual Basic
Public Function GetShort( _
  ByVal columnID As Short _
) As Short

// C#
public short GetShort(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

column value as an Int16.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Cursor class” on page 370
- “Cursor members” on page 370
- “GetColumnID method” on page 395
- “IsNull method” on page 389
- “GetBytes method” on page 374
- “GetBytes method” on page 375
- “GetChars method” on page 377

- "GetBoolean method" on page 373
- "GetDouble method" on page 378
- "GetFloat method" on page 379
- "GetInt method" on page 380
- "GetLong method" on page 381
- "GetString method" on page 383
- "GetTime method" on page 384
- "GetTimestamp method" on page 385
- "GetUInt method" on page 386
- "GetULong method" on page 386
- "GetUShort method" on page 387
- "GetUUID method" on page 388
- "GetColumnSQLType method" on page 397

GetString method

Returns the value for the specified column as a String.

Prototypes

```
' Visual Basic
Public Function GetString( _
  ByVal columnID As Short _
) As String

// C#
public string  GetString(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

column value as a String.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Cursor class" on page 370
- "Cursor members" on page 370
- "GetColumnID method" on page 395
- "IsNull method" on page 389
- "GetBytes method" on page 374
- "GetBytes method" on page 375
- "GetChars method" on page 377
- "GetBoolean method" on page 373
- "GetDouble method" on page 378
- "GetFloat method" on page 379
- "GetInt method" on page 380
- "GetLong method" on page 381

- “GetShort method” on page 382
- “GetTime method” on page 384
- “GetTimestamp method” on page 385
- “GetUInt method” on page 386
- “GetULong method” on page 386
- “GetUShort method” on page 387
- “GetUUID method” on page 388
- “GetColumnSQLType method” on page 397

GetTime method

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

Prototypes

```
' Visual Basic
Public Function GetTime( _
  ByVal columnID As Short _
) As TimeSpan

// C#
public TimeSpan GetTime(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

Column value as a TimeSpan.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Cursor class” on page 370
- “Cursor members” on page 370
- “GetColumnID method” on page 395
- “IsNull method” on page 389
- “GetBytes method” on page 374
- “GetBytes method” on page 375
- “GetChars method” on page 377
- “GetBoolean method” on page 373
- “GetDouble method” on page 378
- “GetFloat method” on page 379
- “GetInt method” on page 380
- “GetLong method” on page 381
- “GetShort method” on page 382
- “GetString method” on page 383
- “GetTimestamp method” on page 385
- “GetUInt method” on page 386

- "GetULong method" on page 386
- "GetUShort method" on page 387
- "GetUUID method" on page 388
- "GetColumnSQLType method" on page 397

GetTimestamp method

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

Prototypes

```
' Visual Basic
Public Function GetTimestamp( _
  ByVal columnID As Short _
) As Date
```

```
// C#
public DateTime GetTimestamp(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

column value as a DateTime.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Cursor class" on page 370
- "Cursor members" on page 370
- DateTime
- "GetColumnID method" on page 395
- "IsNull method" on page 389
- "GetBytes method" on page 374
- "GetBytes method" on page 375
- "GetChars method" on page 377
- "GetBoolean method" on page 373
- "GetDouble method" on page 378
- "GetFloat method" on page 379
- "GetInt method" on page 380
- "GetLong method" on page 381
- "GetShort method" on page 382
- "GetString method" on page 383
- "GetTime method" on page 384
- "GetUInt method" on page 386
- "GetULong method" on page 386
- "GetUShort method" on page 387
- "GetUUID method" on page 388

- "GetColumnSQLType method" on page 397

GetUInt method

Returns the value for the specified column as a UInt32.

Prototypes

```
' Visual Basic
Public Function GetUInt( _
  ByVal columnID As Short _
) As UInt32

// C#
public uint GetUInt(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

column value as an UInt32.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Cursor class" on page 370
- "Cursor members" on page 370
- "GetColumnID method" on page 395
- "IsNull method" on page 389
- "GetBytes method" on page 374
- "GetBytes method" on page 375
- "GetChars method" on page 377
- "GetBoolean method" on page 373
- "GetDouble method" on page 378
- "GetFloat method" on page 379
- "GetInt method" on page 380
- "GetLong method" on page 381
- "GetShort method" on page 382
- "GetString method" on page 383
- "GetTime method" on page 384
- "GetTimestamp method" on page 385
- "GetULong method" on page 386
- "GetUShort method" on page 387
- "GetUUID method" on page 388
- "GetColumnSQLType method" on page 397

GetULong method

Returns the value for the specified column as a UInt64.

Prototypes

```
' Visual Basic
Public Function GetULong( _
  ByVal columnID As Short _
) As UInt64

// C#
public ulong GetULong(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

column value as a UInt64

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Cursor class" on page 370
- "Cursor members" on page 370
- "GetColumnID method" on page 395
- "IsNull method" on page 389
- "GetBytes method" on page 374
- "GetBytes method" on page 375
- "GetChars method" on page 377
- "GetBoolean method" on page 373
- "GetDouble method" on page 378
- "GetFloat method" on page 379
- "GetInt method" on page 380
- "GetLong method" on page 381
- "GetShort method" on page 382
- "GetString method" on page 383
- "GetTime method" on page 384
- "GetTimestamp method" on page 385
- "GetUInt method" on page 386
- "GetUShort method" on page 387
- "GetUUID method" on page 388
- "GetColumnSQLType method" on page 397

GetUShort method

Returns the value for the specified column as a UInt16.

Prototypes

```
' Visual Basic
Public Function GetUShort( _
  ByVal columnID As Short _
) As UInt16
```

```
// C#
public ushort GetUShort(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

column value as an UInt16.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Cursor class” on page 370
- “Cursor members” on page 370
- “GetColumnID method” on page 395
- “IsNull method” on page 389
- “GetBytes method” on page 374
- “GetBytes method” on page 375
- “GetChars method” on page 377
- “GetBoolean method” on page 373
- “GetDouble method” on page 378
- “GetFloat method” on page 379
- “GetInt method” on page 380
- “GetLong method” on page 381
- “GetShort method” on page 382
- “GetString method” on page 383
- “GetTime method” on page 384
- “GetTimestamp method” on page 385
- “GetUInt method” on page 386
- “GetULong method” on page 386
- “GetUUID method” on page 388
- “GetColumnSQLType method” on page 397

GetUUID method

Returns the value for the specified column as a UUID (Guid).

Prototypes

```
' Visual Basic
Public Function GetUUID( _
  ByVal columnID As Short _
) As Guid
```

```
// C#
public Guid GetUUID(
  short columnID
);
```

Parameters

- **columnID** The ID number of the column. The first column in the cursor has an ID value of one.

Return value

Column value as a Guid.

Remarks

Only valid for columns of type SQLType enumeration or for columns of type SQLType enumeration with length 16.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Cursor class" on page 370
- "Cursor members" on page 370
- "GetColumnID method" on page 395
- "IsNull method" on page 389
- "GetBytes method" on page 374
- "GetBytes method" on page 375
- "GetChars method" on page 377
- "GetBoolean method" on page 373
- "GetDouble method" on page 378
- "GetFloat method" on page 379
- "GetInt method" on page 380
- "GetLong method" on page 381
- "GetShort method" on page 382
- "GetTime method" on page 384
- "GetTimestamp method" on page 385
- "GetUInt method" on page 386
- "GetULong method" on page 386
- "GetUShort method" on page 387
- "GetColumnSQLType method" on page 397
- "GetColumnSize method" on page 397

IsNull method

Checks whether the value from the specified column is NULL.

Prototypes

```
' Visual Basic
Public Function IsNull( _
  ByVal columnID As Short _
) As Boolean

// C#
public bool IsNull(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the cursor has an ID value of one.

Return value

true if value is NULL, false if value is not NULL.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Cursor class” on page 370
- “Cursor members” on page 370
- “GetColumnID method” on page 395
- “GetBoolean method” on page 373
- “GetBytes method” on page 374
- “GetBytes method” on page 375
- “GetChars method” on page 377
- “GetDouble method” on page 378
- “GetFloat method” on page 379
- “GetInt method” on page 380
- “GetLong method” on page 381
- “GetShort method” on page 382
- “GetString method” on page 383
- “GetTime method” on page 384
- “GetTimestamp method” on page 385
- “GetUInt method” on page 386
- “GetULong method” on page 386
- “GetUShort method” on page 387
- “GetUUID method” on page 388

MoveAfterLast method

Position to after the last row of the cursor.

Prototypes

```
' Visual Basic
Public Sub MoveAfterLast()

// C#
public void MoveAfterLast();
```

Exceptions

- SQLException class - A SQL error occurred.

MoveBeforeFirst method

Position to before the first row of the cursor.

Prototypes

```
' Visual Basic
Public Sub MoveBeforeFirst()

// C#
public void MoveBeforeFirst();
```

Exceptions

- SQLException class - A SQL error occurred.

MoveFirst method

Position to the first row of the cursor.

Prototypes

' Visual Basic
Public Function **MoveFirst()** As Boolean

// C#
public bool **MoveFirst();**

Return value

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

Exceptions

- SQLException class - A SQL error occurred.

MoveLast method

Position to the last row of the cursor.

Prototypes

' Visual Basic
Public Function **MoveLast()** As Boolean

// C#
public bool **MoveLast();**

Return value

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

Exceptions

- SQLException class - A SQL error occurred.

MoveNext method

Position to the next row or after last.

Prototypes

' Visual Basic
Public Function **MoveNext()** As Boolean

// C#
public bool **MoveNext();**

Return value

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

Exceptions

- SQLException class - A SQL error occurred.

MovePrevious method

Position to the previous row or before first.

Prototypes

' Visual Basic
Public Function **MovePrevious()** As Boolean

// C#
public bool **MovePrevious();**

Return value

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

Exceptions

- SQLException class - A SQL error occurred.

MoveRelative method

Position relative to the current row.

Prototypes

```
' Visual Basic
Public Function MoveRelative( _
  ByVal offset As Integer _
) As Boolean

// C#
public bool MoveRelative(
  int offset
);
```

Parameters

- **offset** The number of rows to move. Negative values correspond to moving backwards.

Return value

true if successful, false otherwise. For example, the method fails if it positions beyond the first or last row.

Remarks

If the row does not exist, the method returns false, and the cursor position is after the last row (`isEOF()`) if *offset* is positive, and before the first row (`isBOF()`) if the *offset* is negative.

Exceptions

- SQLException class - A SQL error occurred.

CursorSchema class

Represents the schema of an UltraLite cursor.

Prototypes

' Visual Basic
MustInherit Public Class **CursorSchema**

// C#
public abstract class **CursorSchema**

Remarks

This class is an abstract base class of the TableSchema class and ResultSetSchema class classes.

CursorSchema members

Public instance properties

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

Public instance methods

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

Protected instance methods

Member	Description
Finalize method	Destructor that cleans up the associated native object.

ColumnCount property

Returns the number of columns in this cursor.

Prototypes

' Visual Basic
Public Readonly Property **ColumnCount** As Short

// C#
public short **ColumnCount** {get;}

Remarks

Column IDs range from 1 to `ColumnCount`, inclusively.

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.

IsOpen property

Checks whether this cursor schema is currently open.

Prototypes

' Visual Basic
Public Readonly Property **IsOpen** As Boolean

// C#
public bool **IsOpen** {get;}

Return value

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

Name property

The name of this cursor.

Prototypes

' Visual Basic
Public Readonly Property **Name** As String

// C#
public string **Name** {get;}

Finalize method

Destructor that cleans up the associated native object.

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

GetColumnID method

Returns the column ID of the named column.

Prototypes

```
' Visual Basic
Public Function GetColumnID( _
  ByVal name As String _
) As Short

// C#
public short GetColumnID(
  string name
);
```

Parameters

- **name** name of the column.

Return value

column ID.

Remarks

Column IDs range from 1 to ColumnCount property, inclusively.

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.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “CursorSchema class” on page 393
- “CursorSchema members” on page 393
- “ColumnCount property” on page 394

GetColumnName method

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

Prototypes

```
' Visual Basic
Public Function GetColumnName( _
  ByVal columnID As Short _
) As String

// C#
public string GetColumnName(
  short columnID
);
```

Parameters

- **columnID** ID of the column. *columnID* must be in the range `[1,CursorSchema.ColumnCount]`.

Return value

name of the column.

Remarks

Column IDs and count may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

Exceptions
- SQLException class - A SQL error occurred.

See also
- "CursorSchema class" on page 393
- "CursorSchema members" on page 393
- "ColumnCount property" on page 394

GetColumnPrecision method

Returns the precision of the named column if the column is a numeric column (SQL type NUMERIC).

Prototypes

```
' Visual Basic
Public Function GetColumnPrecision( _
  ByVal name As String _
) As Integer

// C#
public int GetColumnPrecision(
  string  name
);
```

Parameters
- **name** name of the column.

Return value

precision of the numeric column.

Exceptions
- SQLException class - A SQL error occurred.

See also
- "CursorSchema class" on page 393
- "CursorSchema members" on page 393
- "GetColumnSQLType method" on page 397

GetColumnScale method

Returns the scale of the named column if the column is a numeric column (SQL type NUMERIC).

Prototypes

```
' Visual Basic
Public Function GetColumnScale( _
  ByVal name As String _
) As Integer

// C#
public int GetColumnScale(
  string  name
);
```

Parameters
- **name** name of the column.

Return value

scale of the numeric column.

Exceptions
- SQLException class - A SQL error occurred.

See also

- "CursorSchema class" on page 393
- "CursorSchema members" on page 393
- "GetColumnSQLType method" on page 397

GetColumnSize method

Returns the size of the named column if the column is a sized column (SQL type BINARY or CHAR).

Prototypes

```
' Visual Basic
Public Function GetColumnSize( _
  ByVal name As String _
) As Integer

// C#
public int GetColumnSize(
  string  name
);
```

Parameters

- **name** name of the column.

Return value

size of the sized column.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "CursorSchema class" on page 393
- "CursorSchema members" on page 393
- "GetColumnSQLType method" on page 397

GetColumnSQLType method

Gets the SQL data type of the named column.

Prototypes

```
' Visual Basic
Public Function GetColumnSQLType( _
  ByVal name As String _
) As SQLType

// C#
public SQLType GetColumnSQLType(
  string  name
);
```

Parameters

- **name** name of the column.

Return value

a SQLType enumerated integer.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "CursorSchema class" on page 393
- "CursorSchema members" on page 393

- “SQLType enumeration” on page 478
- “ColumnCount property” on page 394

DatabaseManager class

Manages connections to an UltraLite database. May not be instantiated more than once.

Prototypes

' Visual Basic
NotInheritable Public Class **DatabaseManager**

// C#
public sealed class **DatabaseManager**

Remarks

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

The constructor is not thread-safe. All other methods on this class are thread-safe.

DatabaseManager members

Public instance constructors

Member	Description
DatabaseManager constructor	Initializes the iAnywhere.UltraLite assembly using the standalone runtime. Only one instance of DatabaseManager can be created per application.
DatabaseManager constructor	Initializes the iAnywhere.UltraLite assembly specifying the type of runtime. Only one instance of DatabaseManager can be created per application.

Public instance properties

Member	Description
RuntimeType property	Specifies the type of the UltraLite.NET runtime.

Public instance methods

Member	Description
CreateDatabase method	Creates a database and opens a connection to the database as specified by *parms*.

Member	Description
DropDatabase method	Deletes the specified database. You cannot drop a database that has open connections.
OpenConnection method	Opens a connection to the database specified by *parms*. If the database does not exist, a SQLException class with the value SQLCode enumeration is thrown.
SetActiveSyncListener method	Specifies the listener object to process ActiveSync calls from the MobiLink provider for ActiveSync.
SetServerSyncListener method	Specifies the listener object used to process the specified server sync message.

Protected instance methods

Member	Description
Finalize method	Destructor that cleans up the associated native state.

DatabaseManager constructor

Initializes the iAnywhere.UltraLite assembly using the standalone runtime. Only one instance of DatabaseManager can be created per application.

Prototypes

```
' Visual Basic
Overloads Public Sub New()

// C#
public DatabaseManager();
```

Exceptions

- SQLException class - if called more than once per application.

See also

- "DatabaseManager class" on page 399
- "DatabaseManager members" on page 399
- "RuntimeType property" on page 401

DatabaseManager constructor

Initializes the iAnywhere.UltraLite assembly specifying the type of runtime. Only one instance of DatabaseManager can be created per application.

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal ulType As RuntimeType _
)
```

```
// C#
public DatabaseManager(
  RuntimeType ulType
);
```

Parameters

- **ulType** specifies the type of UltraLite.NET runtime to use.

Exceptions

- SQLException class - if called more than once per application.

See also

- "DatabaseManager class" on page 399
- "DatabaseManager members" on page 399
- "RuntimeType property" on page 401

RuntimeType property

Specifies the type of the UltraLite.NET runtime.

Prototypes

```
' Visual Basic
Public Readonly Property RuntimeType As RuntimeType
```

```
// C#
public RuntimeType RuntimeType {get;}
```

Property value

A RuntimeType enumeration enumerated value representing the type of the unmanaged UltraLite.NET runtime.

CreateDatabase method

Creates a database and opens a connection to the database as specified by *parms*.

Prototypes

```
' Visual Basic
Public Function CreateDatabase( _
  ByVal parms As CreateParms _
) As Connection
```

```
// C#
public Connection CreateDatabase(
  CreateParms parms
);
```

Parameters

- **parms** parameters for creating the database and opening a connection to it. See CreateParms class for more information.

Return value

opened Connection.

Remarks

If the database already exists, a SQLException class with code SQLCode enumeration is thrown.

IsDatabaseNew property on the connection is set to `true` to indicate that the database was created when the connection was opened.

You must close the connection after you have finished all operations on the connection, and before your application terminates. You must close all tables opened on a connection before closing the connection.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "DatabaseManager class" on page 399
- "DatabaseManager members" on page 399
- "CreateParms class" on page 367
- "OpenConnection method" on page 403
- "IsDatabaseNew property" on page 349
- "Close method" on page 353
- "GrantConnectTo method" on page 356
- "RevokeConnectFrom method" on page 358

DropDatabase method

Deletes the specified database.

You cannot drop a database that has open connections.

Prototypes

```
' Visual Basic
Public Sub DropDatabase( _
  ByVal parms As DatabaseNameParms _
)

// C#
public void DropDatabase(
  DatabaseNameParms parms
);
```

Parameters

- **parms** parameters for identifying a database. See DatabaseNameParms class for more information.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "DatabaseManager class" on page 399
- "DatabaseManager members" on page 399
- "DatabaseNameParms class" on page 406
- "CreateDatabase method" on page 401
- "OpenConnection method" on page 403

Finalize method

Destructor that cleans up the associated native state.

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()
```

```
// C#
protected override void Finalize();
```

OpenConnection method

Opens a connection to the database specified by *parms*. If the database does not exist, a SQLException class with the value SQLCode enumeration is thrown.

Prototypes

```
' Visual Basic
Public Function OpenConnection( _
  ByVal parms As ConnectionParms _
) As Connection
```

```
// C#
public Connection OpenConnection(
  ConnectionParms parms
);
```

Parameters

- **parms** parameters for opening a connection. See ConnectionParms class for more information.

Return value

opened Connection.

Remarks

IsDatabaseNew property is set to `false` to indicate that the database was not created when the connection was opened.

You must close the connection after you have finished all operations on the connection, and before your application terminates. You must close all tables opened on a connection before closing the connection.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “DatabaseManager class” on page 399
- “DatabaseManager members” on page 399
- “ConnectionParms class” on page 362
- “CreateDatabase method” on page 401
- “IsDatabaseNew property” on page 349
- “Close method” on page 353
- “GrantConnectTo method” on page 356
- “RevokeConnectFrom method” on page 358

SetActiveSyncListener method

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

Prototypes

```
' Visual Basic
Public Sub SetActiveSyncListener( _
  ByVal appClassName As String, _
  ByVal listener As ActiveSyncListener _
)

// C#
public void SetActiveSyncListener(
  string appClassName,
  ActiveSyncListener listener
);
```

Parameters

- **appClassName** unique class name for the application. This is the class name used when the application is registered for use with ActiveSync.
- **listener** listener object. Use null to remove previous listener.

Remarks

The parameter *appClassName* is the unique identifier used to identify the application. The application may only use one *appClassName* at a time. While a listener is registered with a particular *appClassName*, calls to SetServerSyncListener method or SetActiveSyncListener method with a different *appClassName* will fail.

To remove the ActiveSync listener, call SetActiveSyncListener method with `null` as the *listener* parameter.

To remove all listeners, call SetServerSyncListener method will `null` for all parameters.

Applications should remove all listeners prior to exiting.

Exceptions

- SQLException class - A SQL error occurred.

Example

Refer to the ActiveSyncInvoked method

documentation for an example of SetActiveSyncListener method

.

SetServerSyncListener method

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

Prototypes

```
' Visual Basic
Public Sub SetServerSyncListener( _
  ByVal messageName As String, _
  ByVal appClassName As String, _
  ByVal listener As ServerSyncListener _
)
```

```
// C#
public void SetServerSyncListener(
  string  messageName,
  string  appClassName,
  ServerSyncListener listener
);
```

Parameters

- **messageName** name of the message.
- **appClassName** unique class name for the application. This is a unique identifier used to identify the application.
- **listener** listener object. Use null to remove previous listener.

Remarks

The parameter *appClassName* is the unique identifier used to identify the application. The application may only use one *appClassName* at a time. While a listener is registered with a particular *appClassName*, calls to SetServerSyncListener method or SetActiveSyncListener method with a different *appClassName* will fail.

To remove the listener for a particular message, call SetServerSyncListener method with `null` as the *listener* parameter.

To remove all listeners, call SetServerSyncListener method will `null` for all parameters.

Applications should remove all listeners prior to exiting.

Exceptions

- SQLException class - A SQL error occurred.

Example

Refer to the ServerSyncInvoked method

documentation for an example of SetServerSyncListener method

.

DatabaseNameParms class

Identifies an UltraLite database.

Prototypes

```
' Visual Basic
Public Class DatabaseNameParms
  Inherits Component

// C#
public class DatabaseNameParms :
  Component
```

Remarks

Each instance contains platform-specific and generic paths to the database. Only one value is used with the platform-specific path taking precedence over the generic path. For example, with

```
[Visual Basic]
  dbName.DatabaseOnCE="\UltraLite\mydb1.udb";
  dbName.DatabaseOnDesktop="mydb2.udb";
[C#]
  dbName.DatabaseOnCE="\\UltraLite\\mydb1.udb";
  dbName.DatabaseOnDesktop="mydb2.udb";
```

the path `\UltraLite\mydb1.udb` would be used on Windows CE while `mydb2.udb` would be used on other platforms.

After an instance is passed to an UltraLite method, the property `ParmsUsed` returns the parameters actually used by the UltraLite method.

The recommended extension for UltraLite database files is **.udb**. On Windows CE devices, the default database is **\UltraLiteDB\ulstore.udb**. On other Windows platforms, the default database is **ulstore.udb**. In C#, you must escape any backslash characters in paths.

DatabaseNameParms members

Public instance constructors

Member	Description
DatabaseNameParms constructor	Initializes a DatabaseNameParms instance with its default values.

Public instance properties

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

Member	Description
DatabaseOnCE property	Specifies the path and filename of the UltraLite database on Windows CE.
DatabaseOnDesktop property	Specifies the path and filename of the UltraLite database on Windows desktop platforms.
ParmsUsed property	Returns the parameters actually used by the last `DatabaseManager` method to use this instance.

Public instance methods

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

Public instance events

Member	Description
UnusedEvent event	Unused.

Protected instance methods

Member	Description
Dispose method	Releases the resources used by this object.

DatabaseNameParms constructor

Initializes a DatabaseNameParms instance with its default values.

Prototypes

' Visual Basic
Overloads Public Sub **New()**

// C#
public **DatabaseNameParms();**

Exceptions

- SQLException class - A SQL error occurred.

AdditionalParms property

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

Prototypes

```
' Visual Basic
Public Property AdditionalParms As String
```

```
// C#
public string  AdditionalParms {get;set;}
```

Remarks

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

Additional parameters are:

Keyword	Description
dbn	Identifies a loaded database to which a connection needs to be made. When a database is started, it is assigned a database name, either explicitly with the `dbn` parameter, or by UltraLite using the base of the filename with the extension and path removed. When opening connections, UltraLite will first search for a running database with a matching `dbn`. If one is not found, UltraLite will start a new database using the appropriate database filename parameter (DatabaseOnCE property or DatabaseOnDesktop property). This parameter is required if the application (or UltraLite engine) needs to access two different databases that have the same base filename. This parameter is only used when opening a connection with OpenConnection method or CreateDatabase method.
obfuscate	When set to one (1), specifies that the database is to be created with obfuscation. For example: `createParms.AdditionalParms = "obfuscate=1";` Obfuscating databases provides security against straightforward attempts to view data in the database directly using a viewing tool. It is not proof against skilled and determined attempts to gain access to the data. Obfuscation has little or no performance impact. This parameter is only used when creating a new database with CreateDatabase method.

Keyword	Description
page_size	UltraLite databases are stored in pages. I/O operations are carried out one page at a time. The default page size for UltraLite databases is 4 KB. You can specify 2 KB pages using the following storage parameters string: `connParms.AdditionalParms = "page_ size=2k";` Setting a page size of 2 KB reduces the maximum number of tables to approximately 500. This parameter is only used when creating a new database with CreateDatabase method.
reserve_size	Reserves file system space for storage of UltraLite persistent data. The `reserve_size` parameter allows you to pre-allocate the file system space required for your UltraLite database without actually inserting any data. Reserving file system space can improve performance slightly and also prevent out of memory failures. By default, the persistent storage file only grows when required as the application updates the database. Note that `reserve_size` reserves file system space, which includes the metadata in the persistent store file, and not just the raw data. The metadata overhead, as well as data compression, must be considered when deriving the required file system space from the amount of database data. Running the database with test data and observing the persistent store file size is recommended. The `reserve_size` parameter reserves space by growing the persistent store file to the given reserve size on startup, regardless of whether the file previously existed. The file is never truncated. The following parameter string ensures that the persistent store file is at least 2 MB upon startup. `createParms.AdditionalParms = "reserve_size=2m";` This parameter is only used when opening a connection with OpenConnection method or CreateDatabase method.

DatabaseOnCE property

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

Prototypes

' Visual Basic
Public Property **DatabaseOnCE** As String

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

Remarks

If the value is `null`, the default database **\UltraLiteDB\ulstore.udb** is used. In C#, you must escape any backslash characters in paths.

DatabaseOnDesktop property

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

Prototypes

```
' Visual Basic
Public Property DatabaseOnDesktop As String

// C#
public string  DatabaseOnDesktop {get;set;}
```

Remarks

If the value is `null`, the default database **ulstore.udb** is used. In C#, you must escape any backslash characters in paths.

ParmsUsed property

Returns the parameters actually used by the last `DatabaseManager` method to use this instance.

Prototypes

```
' Visual Basic
Public Readonly Property ParmsUsed As String

// C#
public string  ParmsUsed {get;}
```

See also

- "DatabaseNameParms class" on page 406
- "DatabaseNameParms members" on page 406
- "CreateDatabase method" on page 401
- "DropDatabase method" on page 402
- "OpenConnection method" on page 403

Dispose method

Releases the resources used by this object.

Prototypes

```
' Visual Basic
Overloads Overrides Protected Sub Dispose( _
  ByVal disposing As Boolean _
)

// C#
protected override void Dispose(
  bool disposing
);
```

Parameters

- **disposing** true to release both managed and unmanaged resources; false to release only unmanaged resources.

ToString method

Returns the string representation of this instance.

Prototypes

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

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

Return value

string representation of this instance.

UnusedEvent event

Unused.

Prototypes

' Visual Basic
Public Event **UnusedEvent** As DatabaseNameParms.UnusedEventHandler

// C#
public event DatabaseNameParms.UnusedEventHandler **UnusedEvent**;

Remarks

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

DatabaseNameParms.UnusedEventHandler delegate

Unused.

Prototypes

```
' Visual Basic
 Delegate Sub DatabaseNameParms.UnusedEventHandler( _
   ByVal sender As Object, _
   ByVal args As System.EventArgs _
)

// C#
 delegate void DatabaseNameParms.UnusedEventHandler(
   object sender,
   System.EventArgs args
);
```

Parameters

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

Remarks

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

DatabaseSchema class

Represents the schema of an UltraLite database.

Prototypes

' Visual Basic
NotInheritable Public Class **DatabaseSchema**

// C#
public sealed class **DatabaseSchema**

Remarks

This class cannot be directly instantiated. A DatabaseSchema class object is attached to a connection as its Schema property property and is only valid while that connection is open.

DatabaseSchema members

Public instance properties

Member	Description
CollationName property	The name of the database's collation sequence.
IsCaseSensitive property	Checks whether the database is case sensitive.
IsOpen property	Whether or not the database schema is open.
PublicationCount property	The number of publications in the database.
Signature property	The signature of this database.
TableCount property	The number of tables in the database.

Public instance methods

Member	Description
ApplyFile method	Applies a database schema file to the database while allowing a listener to monitor the progress.
ApplyFile method	Applies a database schema file to the database.
GetDatabaseProperty method	Returns the value of the specified database property.
GetPublicationName method	Returns the name of the publication identified by the specified publication ID. Publication IDs are not publication masks.
GetPublicationSchema method	Returns the publication schema corresponding to the named publication.
GetTableCountInPublications method	Returns the number of tables included in the specified publication mask.

Member	Description
GetTableName method	Returns the name of the table identified by the specified table ID.

CollationName property

The name of the database's collation sequence.

Prototypes

```
' Visual Basic
Public Readonly Property CollationName As String

// C#
public string  CollationName {get;}
```

Remarks

The database collation sequence affects how indexes on tables and result sets are sorted.

IsCaseSensitive property

Checks whether the database is case sensitive.

Prototypes

```
' Visual Basic
Public Readonly Property IsCaseSensitive As Boolean

// C#
public bool IsCaseSensitive {get;}
```

Property value

true if the database is case sensitive, and false if the database is case insensitive.

Remarks

Database case sensitivity affects how indexes on tables and result sets are sorted. Case sensitivity also affects how UserID property and Password property are verified.

IsOpen property

Whether or not the database schema is open.

Prototypes

```
' Visual Basic
Public Readonly Property IsOpen As Boolean

// C#
public bool IsOpen {get;}
```

Property value

True if this database schema is currently open.

PublicationCount property

The number of publications in the database.

Prototypes

' Visual Basic
Public Readonly Property **PublicationCount** As Integer

// C#
public int **PublicationCount** {get;}

Remarks

Publication IDs range from 1 to `PublicationCount`, inclusively. Publication IDs are not publication masks. If the database is not open then PublicationCount has the value 0.

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

See also

- "DatabaseSchema class" on page 413
- "DatabaseSchema members" on page 413
- "GetPublicationName method" on page 418

Signature property

The signature of this database.

Prototypes

' Visual Basic
Public Readonly Property **Signature** As String

// C#
public string **Signature** {get;}

Exceptions

- SQLException class - A SQL error occurred.

TableCount property

The number of tables in the database.

Prototypes

' Visual Basic
Public Readonly Property **TableCount** As Integer

// C#
public int **TableCount** {get;}

Remarks

Table IDs range from 1 to `TableCount`, inclusively. TableCount has the value 0 if the connection is not open.

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.

ApplyFile method

Applies a database schema file to the database while allowing a listener to

monitor the progress.

Prototypes

```
' Visual Basic
Overloads Public Sub ApplyFile( _
  ByVal parms As SchemaParms, _
  ByVal listener As SchemaUpgradeListener _
)

// C#
public void ApplyFile(
  SchemaParms parms,
  SchemaUpgradeListener listener
);
```

Parameters

- **parms** Parameters for specifying the schema to be applied to the database. See SchemaParms class for more information.
- **listener** The listener to receive schema upgrade progress events.

Remarks

All instances of TableSchema class, IndexSchema class, and PublicationSchema class will be invalidated and will need to be replaced.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "DatabaseSchema class" on page 413
- "DatabaseSchema members" on page 413
- "ApplyFile method" on page 416
- "SchemaParms class" on page 458

ApplyFile method

Applies a database schema file to the database.

Prototypes

```
' Visual Basic
Overloads Public Sub ApplyFile( _
  ByVal parms As SchemaParms _
)

// C#
public void ApplyFile(
  SchemaParms parms
);
```

Parameters

- **parms** Parameters for specifying the schema to be applied to the database. See SchemaParms class for more information.

Remarks

All instances of TableSchema class, IndexSchema class, and PublicationSchema class will be invalidated and will need to be replaced.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "DatabaseSchema class" on page 413
- "DatabaseSchema members" on page 413
- "ApplyFile method" on page 415
- "SchemaParms class" on page 458

GetDatabaseProperty method

Returns the value of the specified database property.

Prototypes

```
' Visual Basic
Public Function GetDatabaseProperty( _
  ByVal name As String _
) As String

// C#
public string  GetDatabaseProperty(
  string  name
);
```

Parameters

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

Return value

Value of the property as a string.

Remarks

Recognized properties are:

- `"DATE_FORMAT"` The date format used for string conversions by the database.

  This format is not necessarily the same as the one used by DateTime.
- `"DATE_ORDER"` The date order used for string conversions by the database.
- `"NEAREST_CENTURY"` The nearest century used for string conversions by the database.
- `"PRECISION"` The floating point precision used for string conversions by the database.
- `"SCALE"` The minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION during string conversions by the database.
- `"TIME_FORMAT"` The time format used for string conversions by the database.

  This format is not necessarily the same as the one used by TimeSpan.
- `"TIMESTAMP_FORMAT"` The timestamp format used for string conversions by the database.

  This format is not necessarily the same as the one used by DateTime.

- `"TIMESTAMP_INCREMENT"` The minimum difference between two unique timestamps, in nanoseconds (1,000,000th of a second).

Exceptions

- SQLException class - A SQL error occurred.

GetPublicationName method

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

Prototypes

```
' Visual Basic
Public Function GetPublicationName( _
  ByVal pubID As Integer _
) As String

// C#
public string  GetPublicationName(
  int pubID
);
```

Parameters

- **pubID** ID of the publication. *pubID* must be in the range `[1,DatabaseSchema.PublicationCount]`.

Return value

publication name

Remarks

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

Exceptions

- SQLException class - A SQL error occurred.

See also

- “DatabaseSchema class” on page 413
- “DatabaseSchema members” on page 413
- “PublicationCount property” on page 414

GetPublicationSchema method

Returns the publication schema corresponding to the named publication.

Prototypes

```
' Visual Basic
Public Function GetPublicationSchema( _
  ByVal name As String _
) As PublicationSchema

// C#
public PublicationSchema GetPublicationSchema(
  string  name
);
```

Parameters

- **name** name of the publication.

Return value
publication schema of the named publication.

Exceptions
- SQLException class - A SQL error occurred.

See also
- "DatabaseSchema class" on page 413
- "DatabaseSchema members" on page 413
- "GetPublicationName method" on page 418
- "PublicationSchema class" on page 448

GetTableCountInPublications method

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

Prototypes

```
' Visual Basic
Public Function GetTableCountInPublications( _
  ByVal mask As Integer _
) As Integer
```

```
// C#
public int GetTableCountInPublications(
  int mask
);
```

Parameters
- **mask** set of publications to check.

Return value
number of tables included in set of publications.

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

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

Exceptions
- SQLException class - A SQL error occurred.

GetTableName method

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

Prototypes

```
' Visual Basic
Public Function GetTableName( _
  ByVal tableID As Integer _
) As String
```

```
// C#
public string  GetTableName(
  int tableID
);
```

Parameters
- **tableID** ID of the table. *tableID* must be in range `[1,TableCount]`.

Return value	table name
Remarks	Table IDs may change during a schema upgrade. To correctly identify a table, access it by name or refresh the cached IDs after a schema upgrade.
Exceptions	♦ SQLException class - A SQL error occurred.
See also	♦ “DatabaseSchema class” on page 413 ♦ “DatabaseSchema members” on page 413 ♦ “TableCount property” on page 415

IndexSchema class

Represents the schema of an UltraLite table index.

Prototypes

' Visual Basic
NotInheritable Public Class **IndexSchema**

// C#
public sealed class **IndexSchema**

Remarks

This class cannot be directly instantiated. Index schemas are created using the PrimaryKey property, GetIndex method, and GetOptimalIndex method methods of the TableSchema class class.

IndexSchema members

Public instance properties

Member	Description
ColumnCount property	The number of columns in this index.
IsForeignKey property	Checks whether the index is a foreign key.
IsForeignKeyCheckOnCommit property	Checks whether referential integrity for this foreign key is performed on commits or on inserts and updates.
IsForeignKeyNullable property	Checks whether this foreign key is nullable.
IsOpen property	Determines whether the index schema is open or closed.
IsPrimaryKey property	Checks whether the index is the primary key.
IsUniqueIndex property	Checks whether the index is unique.
IsUniqueKey property	Checks whether the index is a unique key.
Name property	The name of this index.
ReferencedIndexName property	The name of the referenced primary index if this index is a foreign key.
ReferencedTableName property	The name of the referenced primary table if index is a foreign key.

Public instance methods

Member	Description
GetColumnName method	Returns the name of the *colIDInIndex*'th column in this index.
IsColumnDescending method	Checks whether the named column is used in descending order by this index.

Protected instance methods

Member	Description
Finalize method	Cleans up the associated native object.

ColumnCount property

The number of columns in this index.

Prototypes

' Visual Basic
Public Readonly Property **ColumnCount** As Short

// C#
public short **ColumnCount** {get;}

Remarks

Column IDs in indexes range from 1 to `ColumnCount`, inclusively.

Column IDs and count may change during a schema upgrade. Column IDs from an index are different than the column IDs in a table or another index.

IsForeignKey property

Checks whether the index is a foreign key.

Prototypes

' Visual Basic
Public Readonly Property **IsForeignKey** As Boolean

// C#
public bool **IsForeignKey** {get;}

Property value

true if index is the foreign key, false if index is not the foreign key.

Remarks

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

Exceptions

- SQLException class - A SQL error occurred.

IsForeignKeyCheckOnCommit property

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

Prototypes

' Visual Basic
Public Readonly Property **IsForeignKeyCheckOnCommit** As Boolean

// C#
public bool **IsForeignKeyCheckOnCommit** {get;}

Property value — true if referential integrity is checked on commits, false if it is checked on inserts and updates.

Exceptions
- SQLException class - A SQL error occurred.

See also
- "IndexSchema class" on page 421
- "IndexSchema members" on page 421
- "IsForeignKey property" on page 422

IsForeignKeyNullable property

Checks whether this foreign key is nullable.

Prototypes

' Visual Basic
Public Readonly Property **IsForeignKeyNullable** As Boolean

// C#
public bool **IsForeignKeyNullable** {get;}

Property value — true if this foreign key is nullable, false if this foreign key is not nullable.

Exceptions
- SQLException class - A SQL error occurred.

See also
- "IndexSchema class" on page 421
- "IndexSchema members" on page 421
- "IsForeignKey property" on page 422

IsOpen property

Determines whether the index schema is open or closed.

Prototypes

' Visual Basic
Public Readonly Property **IsOpen** As Boolean

// C#
public bool **IsOpen** {get;}

Property value — true if the index schema is open, otherwise closed.

IsPrimaryKey property

Checks whether the index is the primary key.

Prototypes

' Visual Basic
Public Readonly Property **IsPrimaryKey** As Boolean

// C#
public bool **IsPrimaryKey** {get;}

Property value — true if index is the primary key, false if index is not the primary key.

Remarks — Columns in the primary key may not be null.

Exceptions
- SQLException class - A SQL error occurred.

IsUniqueIndex property

Checks whether the index is unique.

Prototypes

```
' Visual Basic
Public Readonly Property IsUniqueIndex As Boolean
```

```
// C#
public bool IsUniqueIndex {get;}
```

Property value — true if index is unique, false if index is not unique.

Remarks — Columns in a unique index may be null.

Exceptions
- SQLException class - A SQL error occurred.

IsUniqueKey property

Checks whether the index is a unique key.

Prototypes

```
' Visual Basic
Public Readonly Property IsUniqueKey As Boolean
```

```
// C#
public bool IsUniqueKey {get;}
```

Property value — true if index is a unique key, false if index is not a unique key.

Remarks — Columns in a unique key may not be null.

Exceptions
- SQLException class - A SQL error occurred.

Name property

The name of this index.

Prototypes

```
' Visual Basic
Public Readonly Property Name As String
```

```
// C#
public string  Name {get;}
```

Exceptions
- SQLException class - A SQL error occurred.

ReferencedIndexName property

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

Prototypes

```
' Visual Basic
Public Readonly Property ReferencedIndexName As String

// C#
public string  ReferencedIndexName {get;}
```

Exceptions

- SQLException class - A SQL error occurred.

ReferencedTableName property

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

Prototypes

```
' Visual Basic
Public Readonly Property ReferencedTableName As String

// C#
public string  ReferencedTableName {get;}
```

Exceptions

- SQLException class - A SQL error occurred.

Finalize method

Cleans up the associated native object.

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()

// C#
protected override void Finalize();
```

GetColumnName method

Returns the name of the *colIDInIndex*'th column in this index.

Prototypes

```
' Visual Basic
Public Function GetColumnName( _
  ByVal colIDInIndex As Short _
) As String

// C#
public string  GetColumnName(
  short colIDInIndex
);
```

Parameters

- **colIDInIndex** ID in this index of the column. *colIDInIndex* must be in the range `[1,IndexSchema.ColumnCount]`.

Return value

name of the column.

Remarks

Column IDs and count may change during a schema upgrade. Column IDs from an index are different than the column IDs in a table or another index.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “IndexSchema class” on page 421
- “IndexSchema members” on page 421
- “ColumnCount property” on page 422

IsColumnDescending method

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

Prototypes

```
' Visual Basic
Public Function IsColumnDescending( _
  ByVal name As String _
) As Boolean

// C#
public bool IsColumnDescending(
  string  name
);
```

Parameters

- **name** name of the column.

Return value

true if the column is used in descending order, false if the column is used in ascending order.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “IndexSchema class” on page 421
- “IndexSchema members” on page 421
- “ColumnCount property” on page 422

PreparedStatement class

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

Prototypes

' Visual Basic
Public Class **PreparedStatement**

// C#
public class **PreparedStatement**

Remarks

This class cannot be directly instantiated. Prepared statements are created using the PrepareStatement method method of the Connection class class. This object can then be used to efficiently execute this statement multiple times.

When a prepared statement is closed, all ResultSet and ResultSetSchema objects associated with it are also closed. PreparedStatement objects are closed when they are garbage collected. However, since all ResultSet and ResultSetSchema objects reference their prepared statement, the prepared statement will not be garbage collected until all its open ResultSet and ResultSetSchema objects are ready for garbage collection.

For resource management reasons, it is recommended that you explicitly close prepared statements when you are done with them.

PreparedStatement members

Public instance properties

Member	Description
HasResultSet property	Indicates whether the prepared statement generates a result set or not.
IsOpen property	Checks whether this prepared statement is currently open.
Plan property	Returns the access plan UltraLite will use to execute a query. This property is intended primarily for use during development.
ResultSetSchema property	Holds the schema describing the result set of this query statement.

Public instance methods

Member	Description
AppendBytesParameter method	Appends the specified subset of the specified array of Bytes (**byte[]** in C#, **Byte()** in Visual Basic) to the new value for the specified SQLType enumeration parameter.

Member	Description
AppendCharsParameter method	Appends the specified subset of the specified array of System.Chars (**char[]** in C#, **Char()** in Visual Basic) to the new value for the specified SQLType enumeration parameter.
Close method	Closes the prepared statement.
ExecuteQuery method	Executes a SQL SELECT statement and returns the result set.
ExecuteStatement method	Executes a statement that does not return a result set, such as a SQL INSERT, DELETE, or UPDATE statement.
SetBoolParameter method	Sets the value for the specified parameter using a Boolean (**bool** in C#).
SetBytesParameter method	Sets the value for the specified parameter using an array of Bytes (**byte[]** in C#, **Byte()** in Visual Basic).
SetDateTimeParameter method	Sets the value for the specified parameter using a DateTime (**Date** in Visual Basic).
SetDoubleParameter method	Sets the value for the specified parameter using a Double (**double** in C#).
SetFloatParameter method	Sets the value for the specified parameter using a Single (**float** in C#).
SetIntParameter method	Sets the value for the specified parameter using an Int32 (**long** in C#, **Long** in Visual Basic).
SetLongParameter method	Sets the value for the specified parameter using an Int64.
SetNullParameter method	Sets a parameter to `NULL`.
SetShortParameter method	Sets the value for the specified parameter using an Int16 (**short** in C#, **Short** in Visual Basic).
SetStringParameter method	Sets the value for the specified parameter using a String.
SetTimeSpanParameter method	Sets the value for the specified parameter using a TimeSpan.
SetUIntParameter method	Sets the value for the specified parameter using a UInt32 (**uint** in C#).
SetULongParameter method	Sets the value for the specified parameter using a UInt64 (**ulong** in C#).
SetUShortParameter method	Sets the value for the specified parameter using a UInt16 (**ushort** in C#).
SetUUIDParameter method	Sets the value for the specified parameter using a UUID (Guid).

Protected instance methods

Member	Description
Finalize method	Finalize method

HasResultSet property

Indicates whether the prepared statement generates a result set or not.

Prototypes

' Visual Basic
Public Readonly Property **HasResultSet** As Boolean

// C#
public bool **HasResultSet** {get;}

Remarks

True if a result set is generated when this statement is executed, false if no result set is generated.

Exceptions

- SQLException class - A SQL error occurred.

IsOpen property

Checks whether this prepared statement is currently open.

Prototypes

' Visual Basic
Public Readonly Property **IsOpen** As Boolean

// C#
public bool **IsOpen** {get;}

Return value

True if this prepared statement is currently open, false if the prepared statement is closed.

Plan property

Returns the access plan UltraLite will use to execute a query. This property is intended primarily for use during development.

Prototypes

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

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

Return value

text-based description of the query execution plan.

Exceptions

- SQLException class - A SQL error occurred.

ResultSetSchema property

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

Prototypes

' Visual Basic
Public Readonly Property **ResultSetSchema** As ResultSetSchema

// C#
public ResultSetSchema **ResultSetSchema** {get;}

Return value

schema of this statement's result set.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "PreparedStatement class" on page 427
- "PreparedStatement members" on page 427
- "HasResultSet property" on page 429
- "ExecuteQuery method" on page 433

AppendBytesParameter method

Appends the specified subset of the specified array of Bytes (**byte[]** in C#, **Byte()** in Visual Basic) to the new value for the specified SQLType enumeration parameter.

Prototypes

```
' Visual Basic
Public Sub AppendBytesParameter( _
  ByVal parameterID As Short, _
  ByVal val As Byte(), _
  ByVal srcOffset As Integer, _
  ByVal count As Integer _
)
```

```
// C#
public void AppendBytesParameter(
  short parameterID,
  byte[] val,
  int srcOffset,
  int count
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the value to append to the current new value for the parameter.
- **srcOffset** start position in the source array.
- **count** the number of bytes to be copied.

Remarks

The bytes at position *srcOffset* (starting from 0) through `srcOffset+count-1` of the array *val* are appended to the value for the specified parameter.

When specifying parameters, the first append to a parameter after an execute clears the current value before appending the new value.

If any of the following are true, a SQLException class with code SQLCode enumeration is thrown and the destination is not modified:

For other errors, a SQLException class with the appropriate error code is thrown.

- *val* is null.
- *srcOffset* is negative.
- *count* is negative.
- `srcOffset+count` is greater than `val.Length`.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "PreparedStatement class" on page 427
- "PreparedStatement members" on page 427
- "SetNullParameter method" on page 440
- "AppendCharsParameter method" on page 431
- "SetBytesParameter method" on page 435
- "SetBoolParameter method" on page 434
- "SetDoubleParameter method" on page 437
- "SetFloatParameter method" on page 438
- "SetIntParameter method" on page 439
- "SetLongParameter method" on page 439
- "SetShortParameter method" on page 441
- "SetStringParameter method" on page 442
- "SetTimeSpanParameter method" on page 443
- "SetDateTimeParameter method" on page 436
- "SetUIntParameter method" on page 444
- "SetULongParameter method" on page 445
- "SetUShortParameter method" on page 446
- "SetUUIDParameter method" on page 446

AppendCharsParameter method

Appends the specified subset of the specified array of System.Chars (**char[]** in C#, **Char()** in Visual Basic) to the new value for the specified SQLType enumeration parameter.

Prototypes

```
' Visual Basic
Public Sub AppendCharsParameter( _
  ByVal parameterID As Short, _
  ByVal val As Char(), _
  ByVal srcOffset As Integer, _
  ByVal count As Integer _
)
```

```
// C#
public void AppendCharsParameter(
  short parameterID,
  char[] val,
  int srcOffset,
  int count
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the value to append to the current value for the parameter.
- **srcOffset** start position in the source array.
- **count** the number of bytes to be copied.

Remarks

The characters at position *srcOffset* (starting from 0) through `srcOffset+count-1` of the array *val* are appended to the value for the specified parameter.

When specifying parameters, the first append to a parameter after an execute clears the current value before appending the new value.

If any of the following are true, a SQLException class with code SQLCode enumeration is thrown and the destination is not modified:

For other errors, a SQLException class with the appropriate error code is thrown.

- *val* is null.
- *srcOffset* is negative.
- *count* is negative.
- `srcOffset+count` is greater than `value.Length`.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “SetNullParameter method” on page 440
- “AppendBytesParameter method” on page 430
- “SetBytesParameter method” on page 435
- “SetBoolParameter method” on page 434
- “SetDoubleParameter method” on page 437
- “SetFloatParameter method” on page 438
- “SetIntParameter method” on page 439
- “SetLongParameter method” on page 439
- “SetShortParameter method” on page 441
- “SetStringParameter method” on page 442

- "SetTimeSpanParameter method" on page 443
- "SetDateTimeParameter method" on page 436
- "SetUIntParameter method" on page 444
- "SetULongParameter method" on page 445
- "SetUShortParameter method" on page 446
- "SetUUIDParameter method" on page 446

Close method

Closes the prepared statement.

Prototypes

```
' Visual Basic
Public Sub Close()

// C#
public void Close();
```

Remarks

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

Exceptions

- SQLException class - A SQL error occurred.

ExecuteQuery method

Executes a SQL SELECT statement and returns the result set.

Prototypes

```
' Visual Basic
Public Function ExecuteQuery() As ResultSet

// C#
public ResultSet ExecuteQuery();
```

Return value

the resulting set of rows from the statement.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "PreparedStatement class" on page 427
- "PreparedStatement members" on page 427
- "HasResultSet property" on page 429
- "ExecuteStatement method" on page 433

ExecuteStatement method

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

Prototypes

```
' Visual Basic
Public Function ExecuteStatement() As Long

// C#
public long ExecuteStatement();
```

Return value — number of rows affected by the statement.

Remarks — If AutoCommit field is true, the statement will be committed only if one or more rows are affected by the statement.

Exceptions
- SQLException class - A SQL error occurred.

See also
- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “HasResultSet property” on page 429
- “ExecuteQuery method” on page 433

Finalize method

Finalize method

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()

// C#
protected override void Finalize();
```

SetBoolParameter method

Sets the value for the specified parameter using a Boolean (**bool** in C#).

Prototypes

```
' Visual Basic
Public Sub SetBoolParameter( _
  ByVal parameterID As Short, _
  ByVal val As Boolean _
)

// C#
public void SetBoolParameter(
  short parameterID,
  bool val
);
```

Parameters
- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Exceptions
- SQLException class - A SQL error occurred.

See also
- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “SetNullParameter method” on page 440
- “AppendBytesParameter method” on page 430
- “AppendCharsParameter method” on page 431

- "SetBytesParameter method" on page 435
- "SetDoubleParameter method" on page 437
- "SetFloatParameter method" on page 438
- "SetIntParameter method" on page 439
- "SetLongParameter method" on page 439
- "SetShortParameter method" on page 441
- "SetStringParameter method" on page 442
- "SetTimeSpanParameter method" on page 443
- "SetDateTimeParameter method" on page 436
- "SetUIntParameter method" on page 444
- "SetULongParameter method" on page 445
- "SetUShortParameter method" on page 446
- "SetUUIDParameter method" on page 446

SetBytesParameter method

Sets the value for the specified parameter using an array of Bytes (**byte[]** in C#, **Byte()** in Visual Basic).

Prototypes

```
' Visual Basic
Public Sub SetBytesParameter( _
  ByVal parameterID As Short, _
  ByVal val As Byte() _
)

// C#
public void SetBytesParameter(
  short parameterID,
  byte[] val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Remarks

Suitable for parameters of type SQLType enumeration or SQLType enumeration only.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "PreparedStatement class" on page 427
- "PreparedStatement members" on page 427
- "SetNullParameter method" on page 440
- "AppendBytesParameter method" on page 430
- "AppendCharsParameter method" on page 431
- "SetBoolParameter method" on page 434
- "SetDoubleParameter method" on page 437

- “SetFloatParameter method” on page 438
- “SetIntParameter method” on page 439
- “SetLongParameter method” on page 439
- “SetShortParameter method” on page 441
- “SetStringParameter method” on page 442
- “SetTimeSpanParameter method” on page 443
- “SetDateTimeParameter method” on page 436
- “SetUIntParameter method” on page 444
- “SetULongParameter method” on page 445
- “SetUShortParameter method” on page 446
- “SetUUIDParameter method” on page 446

SetDateTimeParameter method

Sets the value for the specified parameter using a DateTime (**Date** in Visual Basic).

Prototypes

```
' Visual Basic
Public Sub SetDateTimeParameter( _
  ByVal parameterID As Short, _
  ByVal val As Date _
)

// C#
public void SetDateTimeParameter(
  short parameterID,
  DateTime val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Remarks

The set value is accurate to the millisecond.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “SetNullParameter method” on page 440
- “AppendBytesParameter method” on page 430
- “AppendCharsParameter method” on page 431
- “SetBytesParameter method” on page 435
- “SetBoolParameter method” on page 434
- “SetDoubleParameter method” on page 437
- “SetFloatParameter method” on page 438
- “SetIntParameter method” on page 439

- "SetLongParameter method" on page 439
- "SetShortParameter method" on page 441
- "SetStringParameter method" on page 442
- "SetTimeSpanParameter method" on page 443
- "SetUIntParameter method" on page 444
- "SetULongParameter method" on page 445
- "SetUShortParameter method" on page 446
- "SetUUIDParameter method" on page 446

SetDoubleParameter method

Sets the value for the specified parameter using a Double (**double** in C#).

Prototypes

```
' Visual Basic
Public Sub SetDoubleParameter( _
  ByVal parameterID As Short, _
  ByVal val As Double _
)

// C#
public void SetDoubleParameter(
  short parameterID,
  double val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "PreparedStatement class" on page 427
- "PreparedStatement members" on page 427
- "SetNullParameter method" on page 440
- "AppendBytesParameter method" on page 430
- "AppendCharsParameter method" on page 431
- "SetBytesParameter method" on page 435
- "SetBoolParameter method" on page 434
- "SetFloatParameter method" on page 438
- "SetIntParameter method" on page 439
- "SetLongParameter method" on page 439
- "SetShortParameter method" on page 441
- "SetStringParameter method" on page 442
- "SetTimeSpanParameter method" on page 443
- "SetDateTimeParameter method" on page 436
- "SetUIntParameter method" on page 444

- "SetULongParameter method" on page 445
- "SetUShortParameter method" on page 446
- "SetUUIDParameter method" on page 446

SetFloatParameter method

Sets the value for the specified parameter using a Single (**float** in C#).

Prototypes

```
' Visual Basic
Public Sub SetFloatParameter( _
  ByVal parameterID As Short, _
  ByVal val As Single _
)

// C#
public void SetFloatParameter(
  short parameterID,
  float  val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "PreparedStatement class" on page 427
- "PreparedStatement members" on page 427
- "SetNullParameter method" on page 440
- "AppendBytesParameter method" on page 430
- "AppendCharsParameter method" on page 431
- "SetBytesParameter method" on page 435
- "SetBoolParameter method" on page 434
- "SetDoubleParameter method" on page 437
- "SetIntParameter method" on page 439
- "SetLongParameter method" on page 439
- "SetShortParameter method" on page 441
- "SetStringParameter method" on page 442
- "SetTimeSpanParameter method" on page 443
- "SetDateTimeParameter method" on page 436
- "SetUIntParameter method" on page 444
- "SetULongParameter method" on page 445
- "SetUShortParameter method" on page 446
- "SetUUIDParameter method" on page 446

SetIntParameter method

Sets the value for the specified parameter using an Int32 (**long** in C#, **Long** in Visual Basic).

Prototypes

```
' Visual Basic
Public Sub SetIntParameter( _
  ByVal parameterID As Short, _
  ByVal val As Integer _
)

// C#
public void SetIntParameter(
  short parameterID,
  int val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “SetNullParameter method” on page 440
- “AppendBytesParameter method” on page 430
- “AppendCharsParameter method” on page 431
- “SetBytesParameter method” on page 435
- “SetBoolParameter method” on page 434
- “SetDoubleParameter method” on page 437
- “SetFloatParameter method” on page 438
- “SetLongParameter method” on page 439
- “SetShortParameter method” on page 441
- “SetStringParameter method” on page 442
- “SetTimeSpanParameter method” on page 443
- “SetDateTimeParameter method” on page 436
- “SetUIntParameter method” on page 444
- “SetULongParameter method” on page 445
- “SetUShortParameter method” on page 446
- “SetUUIDParameter method” on page 446

SetLongParameter method

Sets the value for the specified parameter using an Int64.

Prototypes

```
' Visual Basic
Public Sub SetLongParameter( _
  ByVal parameterID As Short, _
  ByVal val As Long _
)

// C#
public void SetLongParameter(
  short parameterID,
  long val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “SetNullParameter method” on page 440
- “AppendBytesParameter method” on page 430
- “AppendCharsParameter method” on page 431
- “SetBytesParameter method” on page 435
- “SetBoolParameter method” on page 434
- “SetDoubleParameter method” on page 437
- “SetFloatParameter method” on page 438
- “SetIntParameter method” on page 439
- “SetShortParameter method” on page 441
- “SetStringParameter method” on page 442
- “SetTimeSpanParameter method” on page 443
- “SetDateTimeParameter method” on page 436
- “SetUIntParameter method” on page 444
- “SetULongParameter method” on page 445
- “SetUShortParameter method” on page 446
- “SetUUIDParameter method” on page 446

SetNullParameter method

Sets a parameter to `NULL`.

Prototypes

```
' Visual Basic
Public Sub SetNullParameter( _
  ByVal parameterID As Short _
)
```

```
// C#
public void SetNullParameter(
  short parameterID
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "PreparedStatement class" on page 427
- "PreparedStatement members" on page 427
- "SetBoolParameter method" on page 434
- "SetDoubleParameter method" on page 437
- "SetFloatParameter method" on page 438
- "SetIntParameter method" on page 439
- "SetLongParameter method" on page 439
- "SetShortParameter method" on page 441
- "SetStringParameter method" on page 442
- "SetTimeSpanParameter method" on page 443
- "SetDateTimeParameter method" on page 436
- "SetUIntParameter method" on page 444
- "SetULongParameter method" on page 445
- "SetUShortParameter method" on page 446
- "SetUUIDParameter method" on page 446

SetShortParameter method

Sets the value for the specified parameter using an Int16 (**short** in C#, **Short** in Visual Basic).

Prototypes

```
' Visual Basic
Public Sub SetShortParameter( _
  ByVal parameterID As Short, _
  ByVal val As Short _
)
```

```
// C#
public void SetShortParameter(
  short parameterID,
  short val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “SetNullParameter method” on page 440
- “AppendBytesParameter method” on page 430
- “AppendCharsParameter method” on page 431
- “SetBytesParameter method” on page 435
- “SetBoolParameter method” on page 434
- “SetDoubleParameter method” on page 437
- “SetFloatParameter method” on page 438
- “SetIntParameter method” on page 439
- “SetLongParameter method” on page 439
- “SetStringParameter method” on page 442
- “SetTimeSpanParameter method” on page 443
- “SetDateTimeParameter method” on page 436
- “SetUIntParameter method” on page 444
- “SetULongParameter method” on page 445
- “SetUShortParameter method” on page 446
- “SetUUIDParameter method” on page 446

SetStringParameter method

Sets the value for the specified parameter using a String.

Prototypes

```
' Visual Basic
Public Sub SetStringParameter( _
  ByVal parameterID As Short, _
  ByVal val As String _
)

// C#
public void SetStringParameter(
  short parameterID,
  string  val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter

Exceptions

- SQLException class - A SQL error occurred.

See also

- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “SetNullParameter method” on page 440
- “AppendBytesParameter method” on page 430
- “AppendCharsParameter method” on page 431

- "SetBytesParameter method" on page 435
- "SetBoolParameter method" on page 434
- "SetDoubleParameter method" on page 437
- "SetFloatParameter method" on page 438
- "SetIntParameter method" on page 439
- "SetLongParameter method" on page 439
- "SetShortParameter method" on page 441
- "SetTimeSpanParameter method" on page 443
- "SetDateTimeParameter method" on page 436
- "SetUIntParameter method" on page 444
- "SetULongParameter method" on page 445
- "SetUShortParameter method" on page 446
- "SetUUIDParameter method" on page 446

SetTimeSpanParameter method

Sets the value for the specified parameter using a TimeSpan.

Prototypes

```
' Visual Basic
Public Sub SetTimeSpanParameter( _
  ByVal parameterID As Short, _
  ByVal val As TimeSpan _
)

// C#
public void SetTimeSpanParameter(
  short parameterID,
  TimeSpan val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Remarks

The set value is accurate to the millisecond and is normalized to a nonnegative value between 0 and 24 hours.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "PreparedStatement class" on page 427
- "PreparedStatement members" on page 427
- "SetNullParameter method" on page 440
- "AppendBytesParameter method" on page 430
- "AppendCharsParameter method" on page 431
- "SetBytesParameter method" on page 435
- "SetBoolParameter method" on page 434
- "SetDoubleParameter method" on page 437

- “SetFloatParameter method” on page 438
- “SetIntParameter method” on page 439
- “SetLongParameter method” on page 439
- “SetShortParameter method” on page 441
- “SetStringParameter method” on page 442
- “SetTimeSpanParameter method” on page 443
- “SetUIntParameter method” on page 444
- “SetULongParameter method” on page 445
- “SetUShortParameter method” on page 446
- “SetUUIDParameter method” on page 446

SetUIntParameter method

Sets the value for the specified parameter using a UInt32 (**uint** in C#).

Prototypes

```
' Visual Basic
Public Sub SetUIntParameter( _
  ByVal parameterID As Short, _
  ByVal val As UInt32 _
)

// C#
public void SetUIntParameter(
  short parameterID,
  uint val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “SetNullParameter method” on page 440
- “AppendBytesParameter method” on page 430
- “AppendCharsParameter method” on page 431
- “SetBytesParameter method” on page 435
- “SetBoolParameter method” on page 434
- “SetDoubleParameter method” on page 437
- “SetFloatParameter method” on page 438
- “SetIntParameter method” on page 439
- “SetLongParameter method” on page 439
- “SetShortParameter method” on page 441
- “SetStringParameter method” on page 442

- "SetTimeSpanParameter method" on page 443
- "SetDateTimeParameter method" on page 436
- "SetULongParameter method" on page 445
- "SetUShortParameter method" on page 446
- "SetUUIDParameter method" on page 446

SetULongParameter method

Sets the value for the specified parameter using a UInt64 (**ulong** in C#).

Prototypes

```
' Visual Basic
Public Sub SetULongParameter( _
  ByVal parameterID As Short, _
  ByVal val As UInt64 _
)

// C#
public void SetULongParameter(
  short parameterID,
  ulong val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "PreparedStatement class" on page 427
- "PreparedStatement members" on page 427
- "SetNullParameter method" on page 440
- "AppendBytesParameter method" on page 430
- "AppendCharsParameter method" on page 431
- "SetBytesParameter method" on page 435
- "SetBoolParameter method" on page 434
- "SetDoubleParameter method" on page 437
- "SetFloatParameter method" on page 438
- "SetIntParameter method" on page 439
- "SetLongParameter method" on page 439
- "SetShortParameter method" on page 441
- "SetStringParameter method" on page 442
- "SetTimeSpanParameter method" on page 443
- "SetDateTimeParameter method" on page 436
- "SetUIntParameter method" on page 444
- "SetUShortParameter method" on page 446
- "SetUUIDParameter method" on page 446

SetUShortParameter method

Sets the value for the specified parameter using a UInt16 (**ushort** in C#).

Prototypes

```
' Visual Basic
Public Sub SetUShortParameter( _
  ByVal parameterID As Short, _
  ByVal val As UInt16 _
)

// C#
public void SetUShortParameter(
  short parameterID,
  ushort val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “SetNullParameter method” on page 440
- “AppendBytesParameter method” on page 430
- “AppendCharsParameter method” on page 431
- “SetBytesParameter method” on page 435
- “SetBoolParameter method” on page 434
- “SetDoubleParameter method” on page 437
- “SetFloatParameter method” on page 438
- “SetIntParameter method” on page 439
- “SetLongParameter method” on page 439
- “SetShortParameter method” on page 441
- “SetStringParameter method” on page 442
- “SetTimeSpanParameter method” on page 443
- “SetDateTimeParameter method” on page 436
- “SetUIntParameter method” on page 444
- “SetULongParameter method” on page 445
- “SetUUIDParameter method” on page 446

SetUUIDParameter method

Sets the value for the specified parameter using a UUID (Guid).

Prototypes

```
' Visual Basic
Public Sub SetUUIDParameter( _
  ByVal parameterID As Short, _
  ByVal val As Guid _
)

// C#
public void SetUUIDParameter(
  short parameterID,
  Guid val
);
```

Parameters

- **parameterID** the ID number of the parameter. The first parameter in the statement has an ID value of one.
- **val** the new value for the parameter.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “PreparedStatement class” on page 427
- “PreparedStatement members” on page 427
- “GetNewUUID method” on page 356
- “SetNullParameter method” on page 440
- “AppendBytesParameter method” on page 430
- “AppendCharsParameter method” on page 431
- “SetBytesParameter method” on page 435
- “SetBoolParameter method” on page 434
- “SetDoubleParameter method” on page 437
- “SetFloatParameter method” on page 438
- “SetIntParameter method” on page 439
- “SetLongParameter method” on page 439
- “SetShortParameter method” on page 441
- “SetTimeSpanParameter method” on page 443
- “SetDateTimeParameter method” on page 436
- “SetUIntParameter method” on page 444
- “SetULongParameter method” on page 445
- “SetUShortParameter method” on page 446

PublicationSchema class

Represents the schema of an UltraLite publication.

Prototypes

' Visual Basic
NotInheritable Public Class **PublicationSchema**

// C#
public sealed class **PublicationSchema**

Remarks

This class cannot be directly instantiated. Publication schemas are created using the GetPublicationSchema method method of the DatabaseSchema class class.

UltraLite methods requiring a publication mask actually require a set of publications to check. A set is formed by or'ing the publication masks of individual publications. For example:

```
pub1.Mask | pub2.Mask
```

Two special mask values are also provided by this class. SYNC_ALL_DB field corresponds to the entire database. SYNC_ALL_PUBS field corresponds to all publications.

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

See also

- "PublicationSchema members" on page 448
- "Mask property" on page 449
- "GetLastDownloadTime method" on page 355
- "CountUploadRows method" on page 354
- "Schema property" on page 351

PublicationSchema members

Public static fields (Shared)

Member	Description
SYNC_ALL_DB field	Publication mask corresponding to the entire database.
SYNC_ALL_PUBS field	Publication mask corresponding to all publications.

Public instance properties

Member	Description
IsOpen property	Determines whether the publication schema is open or closed.

Member	Description
Mask property	The publication mask of this publication.
Name property	The name of this publication.

SYNC_ALL_DB field

Publication mask corresponding to the entire database.

Prototypes

' Visual Basic
Public Shared **SYNC_ALL_DB** As Integer

// C#
public const int **SYNC_ALL_DB**;

SYNC_ALL_PUBS field

Publication mask corresponding to all publications.

Prototypes

' Visual Basic
Public Shared **SYNC_ALL_PUBS** As Integer

// C#
public const int **SYNC_ALL_PUBS**;

IsOpen property

Determines whether the publication schema is open or closed.

Prototypes

' Visual Basic
Public Readonly Property **IsOpen** As Boolean

// C#
public bool **IsOpen** {get;}

Property value

True if the publication schema is open, otherwise closed.

Mask property

The publication mask of this publication.

Prototypes

' Visual Basic
Public Readonly Property **Mask** As Integer

// C#
public int **Mask** {get;}

Remarks

Publication IDs, masks, and counts may change during a schema upgrade.

To correctly identify a publication, access it by name or refresh the cached masks, and counts after a schema upgrade.

Exceptions

- SQLException class - If a SQL error exception occurs. PUBLICATION_NOT_FOUND is issued if a schema upgrade has deleted or renamed this publication.

Name property

The name of this publication.

Prototypes

```
' Visual Basic
Public Readonly Property Name As String

// C#
public string  Name {get;}
```

Exceptions

- SQLException class - A SQL error occurred. PUBLICATION_NOT_FOUND is issued if a schema upgrade has deleted or renamed this publication.

ResultSet class

Represents a result set in an UltraLite database. Created at runtime using ExecuteQuery method.

Prototypes

```
' Visual Basic
Public Class ResultSet
  Inherits Cursor
```

```
// C#
public class ResultSet :
  Cursor
```

Remarks

This class cannot be directly instantiated. Result sets are created using the ExecuteQuery method method of the PreparedStatement class class.

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

ResultSet members

Public instance properties

Member	Description
IsBOF property (inherited from Cursor)	Returns true if the current row position is before the first row.
IsEOF property (inherited from Cursor)	Returns true if the current row position is after the last row.
IsOpen property (inherited from Cursor)	Checks whether this cursor is currently open.
RowCount property (inherited from Cursor)	Returns the number of rows in the cursor.
Schema property	Holds the schema of this result set. This property is only valid while its prepared statement is open.

Public instance methods

Member	Description
Close method	Closes the result set.
Fill method (inherited from Cursor)	Fills a ADO .NET DataTable object with the rows in the cursor.
GetBoolean method (inherited from Cursor)	Returns the value for the specified column as a Boolean.

Member	Description
GetBytes method (inherited from Cursor)	Returns the value for the specified column as an array of Bytes. Only valid for columns of type SQLType enumeration, SQLType enumeration, or SQLType enumeration.
GetBytes method (inherited from Cursor)	Copies a subset of the value for the specified SQLType enumeration column, beginning at the specified offset, to the specified offset of the destination Byte array.
GetChars method (inherited from Cursor)	Copies a subset of the value for the specified SQLType enumeration column, beginning at the specified offset, to the specified offset of the destination System.Char array.
GetDouble method (inherited from Cursor)	Returns the value for the specified column as a Double.
GetFloat method (inherited from Cursor)	Returns the value for the specified column as a Single.
GetInt method (inherited from Cursor)	Returns the value for the specified column as an Int32.
GetLong method (inherited from Cursor)	Returns the value for the specified column as an Int64.
GetShort method (inherited from Cursor)	Returns the value for the specified column as an Int16.
GetString method (inherited from Cursor)	Returns the value for the specified column as a String.
GetTime method (inherited from Cursor)	Returns the value for the specified column as a TimeSpan with millisecond accuracy.
GetTimestamp method (inherited from Cursor)	Returns the value for the specified column as a DateTime with millisecond accuracy.
GetUInt method (inherited from Cursor)	Returns the value for the specified column as a UInt32.
GetULong method (inherited from Cursor)	Returns the value for the specified column as a UInt64.
GetUShort method (inherited from Cursor)	Returns the value for the specified column as a UInt16.
GetUUID method (inherited from Cursor)	Returns the value for the specified column as a UUID (Guid).

Member	Description
IsNull method (inherited from Cursor)	Checks whether the value from the specified column is NULL.
MoveAfterLast method (inherited from Cursor)	Position to after the last row of the cursor.
MoveBeforeFirst method (inherited from Cursor)	Position to before the first row of the cursor.
MoveFirst method (inherited from Cursor)	Position to the first row of the cursor.
MoveLast method (inherited from Cursor)	Position to the last row of the cursor.
MoveNext method (inherited from Cursor)	Position to the next row or after last.
MovePrevious method (inherited from Cursor)	Position to the previous row or before first.
MoveRelative method (inherited from Cursor)	Position relative to the current row.

Protected instance methods

Member	Description
Finalize method	Finalize method

Schema property

Holds the schema of this result set. This property is only valid while its prepared statement is open.

Prototypes

' Visual Basic
Public Readonly Property **Schema** As ResultSetSchema

// C#
public ResultSetSchema **Schema** {get;}

See also

- "ResultSet class" on page 451
- "ResultSet members" on page 451
- "ResultSetSchema class" on page 455

Close method

Closes the result set.

Prototypes

' Visual Basic
NotOverridable Public Sub **Close()**

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

Remarks

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

Exceptions

- SQLException class - A SQL error occurred.

Finalize method

Finalize method

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

ResultSetSchema class

Represents the schema of an UltraLite result set.

Prototypes

' Visual Basic
NotInheritable Public Class **ResultSetSchema**
  Inherits CursorSchema

// C#
public sealed class **ResultSetSchema** :
  CursorSchema

Remarks

This class cannot be directly instantiated. A ResultSetSchema class object is attached to a result set as its Schema property property. A result set schema is also attached to a prepared statement as its ResultSetSchema property property if the prepared statement represents a query (when the prepared statement's HasResultSet property property is true).

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

See also

- "ResultSetSchema members" on page 455
- "PreparedStatement class" on page 427
- "ResultSet class" on page 451

ResultSetSchema members

Public instance properties

Member	Description
ColumnCount property (inherited from CursorSchema)	Returns the number of columns in this cursor.
IsOpen property (inherited from CursorSchema)	Checks whether this cursor schema is currently open.
Name property	The name of this cursor.

Public instance methods

Member	Description
GetColumnID method (inherited from CursorSchema)	Returns the column ID of the named column.
GetColumnName method (inherited from CursorSchema)	Returns the name of column identified by the specified column ID.

Member	Description
GetColumnPrecision method (inherited from CursorSchema)	Returns the precision of the named column if the column is a numeric column (SQL type NUMERIC).
GetColumnScale method (inherited from CursorSchema)	Returns the scale of the named column if the column is a numeric column (SQL type NUMERIC).
GetColumnSize method (inherited from CursorSchema)	Returns the size of the named column if the column is a sized column (SQL type BINARY or CHAR).
GetColumnSQLType method (inherited from CursorSchema)	Gets the SQL data type of the named column.

Protected instance methods

Member	Description
Finalize method	Destructor which cleans up the associated native object.

Name property

The name of this cursor.

Prototypes

' Visual Basic
Public Readonly Property **Name** As String

// C#
public string **Name** {get;}

Return value

Sql statement that generated this ResultSetSchema.

Finalize method

Destructor which cleans up the associated native object.

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

RuntimeType enumeration

Enumerates the types of UltraLite.NET runtimes.

Prototypes

' Visual Basic
Public Enum **RuntimeType**

// C#
public enum **RuntimeType**

Members

Member	Description
STANDALONE_UL	Selects the standalone UltraLite.NET runtime. The standalone runtime accesses databases directly. Databases are accessed more quickly but cannot be shared.
UL_ENGINE_CLIENT	Selects to use the UltraLite engine. The UltraLite.NET engine client communicates with the UltraLite engine to access databases. This means that databases can be shared by different applications.

See also

- "DatabaseManager constructor" on page 400

SchemaParms class

Specifies the schema for an UltraLite database.

Prototypes

```
' Visual Basic
NotInheritable Public Class SchemaParms
  Inherits Component
```

```
// C#
public sealed class SchemaParms :
  Component
```

Remarks

Each instance contains platform-specific and generic paths to a schema file. Only one file value is used with the platform-specific keyword taking precedence over the generic keyword. For example, with `schemaParms.SchemaOnCE="\\s1.usm";` and `schemaParms.SchemaOnDesktop="s2.usm";`, the path `\s1.usm` would be used on Windows CE while `s2.usm` would be used on other platforms.

After an instance is passed to an UltraLite method, the property `ParmsUsed` returns the parameters actually used by the UltraLite method.

The recommended extension for UltraLite schema files is **.usm**. You must escape any backslash characters in paths.

SchemaParms members

Public instance constructors

Member	Description
SchemaParms constructor	Initializes a SchemaParms instance with its default values.
SchemaParms constructor	Initializes a SchemaParms instance with the specified values.

Public instance properties

Member	Description
AdditionalParms property	Specifies additional parameters as a semicolon-separated list of *name=value* pairs. This is reserved for future use.
ParmsUsed property	Returns the parameters actually used by the last `DatabaseSchema` method to use this instance.
SchemaOnCE property	Specifies the path and filename of the UltraLite schema on Windows CE.

Member	Description
SchemaOnDesktop property	Specifies the path and filename of the UltraLite schema on Windows desktop platforms.

Public instance methods

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

Protected instance methods

Member	Description
Finalize method	Cleans up the associated native object.

SchemaParms constructor

Initializes a SchemaParms instance with its default values.

Prototypes

```
' Visual Basic
Overloads Public Sub New()

// C#
public SchemaParms();
```

Exceptions

- SQLException class - A SQL error occurred.

SchemaParms constructor

Initializes a SchemaParms instance with the specified values.

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal schemaOnCE As String, _
  ByVal schemaOnDesktop As String, _
  ByVal additionalParms As String _
)

// C#
public SchemaParms(
  string schemaOnCE,
  string schemaOnDesktop,
  string additionalParms
);
```

Parameters

- **schemaOnCE** value for SchemaOnCE property.
- **schemaOnDesktop** value for SchemaOnDesktop property.
- **additionalParms** value for AdditionalParms property.

Exceptions

- SQLException class - A SQL error occurred.

AdditionalParms property

Specifies additional parameters as a semicolon-separated list of *name=value* pairs. This is reserved for future use.

Prototypes

' Visual Basic
Public Property **AdditionalParms** As String

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

ParmsUsed property

Returns the parameters actually used by the last `DatabaseSchema` method to use this instance.

Prototypes

' Visual Basic
Public Readonly Property **ParmsUsed** As String

// C#
public string **ParmsUsed** {get;}

See also

- "SchemaParms class" on page 458
- "SchemaParms members" on page 458
- "ApplyFile method" on page 415

SchemaOnCE property

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

Prototypes

' Visual Basic
Public Property **SchemaOnCE** As String

// C#
public string **SchemaOnCE** {get;set;}

SchemaOnDesktop property

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

Prototypes

' Visual Basic
Public Property **SchemaOnDesktop** As String

// C#
public string **SchemaOnDesktop** {get;set;}

Finalize method

Cleans up the associated native object.

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

ToString method

Returns the string representation of this instance.

Prototypes

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

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

Return value

string representation of this instance.

SchemaUpgradeData class

Returns schema upgrade monitoring data.

Prototypes

' Visual Basic
Public Class **SchemaUpgradeData**

// C#
public class **SchemaUpgradeData**

See also

- "SchemaUpgradeData members" on page 462
- "SchemaUpgradeListener interface" on page 464

SchemaUpgradeData members

Public instance properties

Member	Description
FinalProgressCount property	Estimate of the final progress count.
ProgressCounter property	The schema upgrade progress counter. This number is an approximation of the progess so far.
State property	The current schema upgrade state.
UpgradeOperations property	Number of schema upgrade operations performed.

FinalProgressCount property

Estimate of the final progress count.

Prototypes

' Visual Basic
Public Readonly Property **FinalProgressCount** As Long

// C#
public long **FinalProgressCount** {get;}

Return value

estimated final progress count.

See also

- "SchemaUpgradeData class" on page 462
- "SchemaUpgradeData members" on page 462
- "SchemaUpgradeState enumeration" on page 465
- "ProgressCounter property" on page 462

ProgressCounter property

The schema upgrade progress counter. This number is an approximation of

the progess so far.

Prototypes

' Visual Basic
Public Readonly Property **ProgressCounter** As Long

// C#
public long **ProgressCounter** {get;}

Return value

schema upgrade progress counter as a number between 0 and FinalProgressCount property.

See also

- "SchemaUpgradeData class" on page 462
- "SchemaUpgradeData members" on page 462
- "SchemaUpgradeState enumeration" on page 465

State property

The current schema upgrade state.

Prototypes

' Visual Basic
Public Readonly Property **State** As SchemaUpgradeState

// C#
public SchemaUpgradeState **State** {get;}

See also

- "SchemaUpgradeData class" on page 462
- "SchemaUpgradeData members" on page 462
- "SchemaUpgradeState enumeration" on page 465

UpgradeOperations property

Number of schema upgrade operations performed.

Prototypes

' Visual Basic
Public Readonly Property **UpgradeOperations** As Long

// C#
public long **UpgradeOperations** {get;}

Remarks

This number is an approximation of the amount of work done during the schema upgrade. This number starts at zero and increases as the update proceeds. This number is updated more frequently than the progress counter and can be used as a relative measure to compare against other schema upgrades.

See also

- "SchemaUpgradeData class" on page 462
- "SchemaUpgradeData members" on page 462
- "SchemaUpgradeState enumeration" on page 465

SchemaUpgradeListener interface

The listener interface for receiving schema upgrade progress events.

Prototypes

```
' Visual Basic
Public Interface SchemaUpgradeListener
```

```
// C#
public interface SchemaUpgradeListener
```

See also

- "SchemaUpgradeListener members" on page 464
- "ApplyFile method" on page 415

SchemaUpgradeListener members

Public instance methods

Member	Description
SchemaUpgrading method	Invoked during schema upgrade to inform the user of progress. When the state is SchemaUpgradeState enumeration, this method should return true to cancel the schema upgrade or return false to continue. For other states, this method's return value is ignored.

SchemaUpgrading method

Invoked during schema upgrade to inform the user of progress. When the state is SchemaUpgradeState enumeration, this method should return true to cancel the schema upgrade or return false to continue. For other states, this method's return value is ignored.

Prototypes

```
' Visual Basic
Public Function SchemaUpgrading( _
  ByVal data As SchemaUpgradeData _
) As Boolean
```

```
// C#
public bool SchemaUpgrading(
  SchemaUpgradeData data
);
```

Parameters

- **data** object containing the latest schema upgrade progress data.

Return value

This method should return false to continue or true to cancel the schema upgrade (state must be SchemaUpgradeState enumeration to cancel).

Remarks

No UltraLite.NET API methods should be invoked during a `SchemaUpgrading` call.

SchemaUpgradeState enumeration

Enumerates all the states that can occur while upgrading a schema.

Prototypes

' Visual Basic
Public Enum **SchemaUpgradeState**

// C#
public enum **SchemaUpgradeState**

Remarks

STATE_STARTING is the only state during which the upgrade may be cancelled. If the upgrade is cancelled, you will receive a second event with state **STATE_ERROR**.

Members

Member	Description
STATE_ABORT	The schema upgrade has been aborted and the old database is preserved. This state may occur as the result of a recoverable error or user abort.
STATE_DONE	The schema upgrade completed successfully.
STATE_ERROR	A critial error occured and the database is unusable.
STATE_LAST	Internally used state not sent to SchemaUpgrading method.
STATE_STARTING	The schema upgrade is starting. This is the only state during which the upgrade may be cancelled. If the upgrade is cancelled, you will receive a second event with state **STATE_ABORT**.
STATE_UPGRADING	The schema upgrade is in progress.

See also

- "State property" on page 463
- "ProgressCounter property" on page 462
- "FinalProgressCount property" on page 462
- "UpgradeOperations property" on page 463

ServerSyncListener interface

The listener interface for receiving server synchronization messages.

Prototypes

```
' Visual Basic
Public Interface ServerSyncListener
```

```
// C#
public interface ServerSyncListener
```

ServerSyncListener members

Public instance methods

Member	Description
ServerSyncInvoked method	Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

ServerSyncInvoked method

Invoked when the MobiLink Listener for server-initiated synchronizations calls the application to perform synchronization.

Prototypes

```
' Visual Basic
Public Sub ServerSyncInvoked( _
  ByVal messageName As String _
)
```

```
// C#
public void ServerSyncInvoked(
  string  messageName
);
```

Parameters

- **messageName** name of the message sent to the application.

Remarks

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

Example

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

```
[Visual Basic]
Imports iAnywhere.UltraLite

              Public Class MainWindow
  Inherits System.Windows.Forms.Form
  Implements ServerSyncListener
  Private dbMgr As DatabaseManager
  Private conn As Connection

            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
    dbMgr = New DatabaseManager
    dbMgr.SetServerSyncListener( "myCompany.mymsg",
"myCompany.myapp", Me )
    'Create Connection
    ...
  End Sub

            Protected Overrides Sub
OnClosing(ByVal e As
System.ComponentModel.CancelEventArgs)
    If Not (dbMgr Is Nothing) Then
      dbMgr.SetServerSyncListener(Nothing, Nothing,
Nothing)
      dbMgr = Nothing
    End If
  End Sub

            Public Sub ServerSyncInvoked(ByVal
messageName As String) Implements
iAnywhere.UltraLite.ServerSyncListener.ServerSyncInvoke
d
    Me.Invoke(New EventHandler(AddressOf
Me.ServerSyncAction))
  End Sub

            Public Sub ServerSyncAction(ByVal
sender As Object, ByVal e As EventArgs)
    ' Do Server sync
    conn.Synchronize()
  End Sub
End Class

          [C#]
using iAnywhere.UltraLite;
public class Form1 : System.Windows.Forms.Form,
ServerSyncListener
{
  private System.Windows.Forms.MainMenu mainMenu1;
  private DatabaseManager dbMgr;
  private Connection conn;

            public Form1()
```

SQLCode enumeration

Enumerates the SQL codes that may be reported by UltraLite.NET.

Prototypes

' Visual Basic
Public Enum **SQLCode**

// C#
public enum **SQLCode**

Members

Member	Description
SQLE_AGGREGATES_NOT_-ALLOWED	SQLE_AGGREGATES_NOT_ALLOWED(-150)
SQLE_ALIAS_NOT_UNIQUE	SQLE_ALIAS_NOT_UNIQUE(-830)
SQLE_ALIAS_NOT_YET_-DEFINED	SQLE_ALIAS_NOT_YET_DEFINED(-831)
SQLE_AMBIGUOUS_INDEX_-NAME	SQLE_AMBIGUOUS_INDEX_NAME(-678)
SQLE_ARGUMENT_-CANNOT_BE_NULL	SQLE_ARGUMENT_CANNOT_BE_NULL(-90)
SQLE_BAD_ENCRYPTION_-KEY	SQLE_BAD_ENCRYPTION_KEY(-840)
SQLE_BAD_PARAM_INDEX	SQLE_BAD_PARAM_INDEX(-689)
SQLE_CANNOT_ACCESS_-FILE	SQLE_CANNOT_ACCESS_FILE(-602)
SQLE_CANNOT_ACCESS_-SCHEMA_FILE	SQLE_CANNOT_ACCESS_SCHEMA_FILE(-951)
SQLE_CANNOT_CHANGE_-USER_NAME	SQLE_CANNOT_CHANGE_USER_NAME(-867)
SQLE_CANNOT_EXECUTE_-STMT	SQLE_CANNOT_EXECUTE_STMT(111)
SQLE_CANNOT_MODIFY	SQLE_CANNOT_MODIFY(-191)
SQLE_CANNOT_REGISTER_-LISTENER	SQLE_CANNOT_REGISTER_LISTENER(-992)
SQLE_COLUMN_-AMBIGUOUS	SQLE_COLUMN_AMBIGUOUS(-144)

Member	Description
SQLE_COLUMN_CANNOT_-BE_NULL	SQLE_COLUMN_CANNOT_BE_NULL(-195)
SQLE_COLUMN_IN_INDEX	SQLE_COLUMN_IN_INDEX(-127)
SQLE_COLUMN_NOT_-FOUND	SQLE_COLUMN_NOT_FOUND(-143)
SQLE_COLUMN_NOT_-FOUND_IN_TABLE	SQLE_COLUMN_NOT_FOUND_IN_TABLE(-834)
SQLE_COMMUNICATIONS_-ERROR	SQLE_COMMUNICATIONS_ERROR(-85)
SQLE_CONNECTION_-ALREADY_EXISTS	SQLE_CONNECTION_ALREADY_EXISTS(-955)
SQLE_CONNECTION_NOT_-FOUND	SQLE_CONNECTION_NOT_FOUND(-108)
SQLE_CONNECTION_-RESTORED	SQLE_CONNECTION_RESTORED(133)
SQLE_CONVERSION_ERROR	SQLE_CONVERSION_ERROR(-157)
SQLE_CORRELATION_-NAME_NOT_FOUND	SQLE_CORRELATION_NAME_NOT_FOUND(-142)
SQLE_COULD_NOT_FIND_-FUNCTION	SQLE_COULD_NOT_FIND_FUNCTION(-621)
SQLE_COULD_NOT_LOAD_-LIBRARY	SQLE_COULD_NOT_LOAD_LIBRARY(-620)
SQLE_CURSOR_ALREADY_-OPEN	SQLE_CURSOR_ALREADY_OPEN(-172)
SQLE_CURSOR_NOT_OPEN	SQLE_CURSOR_NOT_OPEN(-180)
SQLE_CURSOR_RESTORED	SQLE_CURSOR_RESTORED(134)
SQLE_CURSOROP_NOT_-ALLOWED	SQLE_CURSOROP_NOT_ALLOWED(-187)
SQLE_DATABASE_CREATED	SQLE_DATABASE_CREATED(136)
SQLE_DATABASE_ERROR	SQLE_DATABASE_ERROR(-301)
SQLE_DATABASE_NAME_-REQUIRED	SQLE_DATABASE_NAME_REQUIRED(-87)

Member	Description
SQLE_DATABASE_NEW	SQLE_DATABASE_NEW(123)
SQLE_DATABASE_NOT_-CREATED	SQLE_DATABASE_NOT_CREATED(-645)
SQLE_DATABASE_-UPGRADE_NOT_POSSIBLE	SQLE_DATABASE_UPGRADE_NOT_POSSIBLE(-673)
SQLE_DATATYPE_NOT_-ALLOWED	SQLE_DATATYPE_NOT_ALLOWED(-624)
SQLE_DBSPACE_FULL	SQLE_DBSPACE_FULL(-604)
SQLE_DEVICE_IO_FAILED	SQLE_DEVICE_IO_FAILED(-974)
SQLE_DIV_ZERO_ERROR	SQLE_DIV_ZERO_ERROR(-628)
SQLE_DOWNLOAD_-CONFLICT	SQLE_DOWNLOAD_CONFLICT(-839)
SQLE_DROP_DATABASE_-FAILED	SQLE_DROP_DATABASE_FAILED(-651)
SQLE_DYNAMIC_MEMORY_-EXHAUSTED	SQLE_DYNAMIC_MEMORY_EXHAUSTED(-78)
SQLE_ENGINE_ALREADY_-RUNNING	SQLE_ENGINE_ALREADY_RUNNING(-96)
SQLE_ENGINE_NOT_-MULTIUSER	SQLE_ENGINE_NOT_MULTIUSER(-89)
SQLE_ERROR	SQLE_ERROR(-300)
SQLE_ERROR_CALLING_-FUNCTION	SQLE_ERROR_CALLING_FUNCTION(-622)
SQLE_EXPRESSION_ERROR	SQLE_EXPRESSION_ERROR(-156)
SQLE_FOREIGN_KEY_-NAME_NOT_FOUND	SQLE_FOREIGN_KEY_NAME_NOT_FOUND(-145)
SQLE_IDENTIFIER_TOO_-LONG	SQLE_IDENTIFIER_TOO_LONG(-250)
SQLE_INCORRECT_-VOLUME_ID	SQLE_INCORRECT_VOLUME_ID(-975)
SQLE_INDEX_NAME_NOT_-UNIQUE	SQLE_INDEX_NAME_NOT_UNIQUE(-111)

Member	Description
SQLE_INDEX_NOT_FOUND	SQLE_INDEX_NOT_FOUND(-183)
SQLE_INDEX_NOT_UNIQUE	SQLE_INDEX_NOT_UNIQUE(-196)
SQLE_INTERRUPTED	SQLE_INTERRUPTED(-299)
SQLE_INVALID_DATABASE	SQLE_INVALID_DATABASE(-84)
SQLE_INVALID_FOREIGN_-KEY	SQLE_INVALID_FOREIGN_KEY(-194)
SQLE_INVALID_FOREIGN_-KEY_DEF	SQLE_INVALID_FOREIGN_KEY_DEF(-113)
SQLE_INVALID_GROUP_-SELECT	SQLE_INVALID_GROUP_SELECT(-149)
SQLE_INVALID_LOGON	SQLE_INVALID_LOGON(-103)
SQLE_INVALID_OPTION_-SETTING	SQLE_INVALID_OPTION_SETTING(-201)
SQLE_INVALID_ORDER	SQLE_INVALID_ORDER(-152)
SQLE_INVALID_PARAMETER	SQLE_INVALID_PARAMETER(-735)
SQLE_INVALID_PARSE_-PARAMETER	SQLE_INVALID_PARSE_PARAMETER(-95)
SQLE_INVALID_SQL_-IDENTIFIER	SQLE_INVALID_SQL_IDENTIFIER(-760)
SQLE_LOCKED	SQLE_LOCKED(-210)
SQLE_MEMORY_ERROR	SQLE_MEMORY_ERROR(-309)
SQLE_METHOD_CANNOT_-BE_CALLED	SQLE_METHOD_CANNOT_BE_CALLED(-669)
SQLE_NAME_NOT_UNIQUE	SQLE_NAME_NOT_UNIQUE(-110)
SQLE_NO_COLUMN_NAME	SQLE_NO_COLUMN_NAME(-163)
SQLE_NO_CURRENT_ROW	SQLE_NO_CURRENT_ROW(-197)
SQLE_NO_INDICATOR	SQLE_NO_INDICATOR(-181)
SQLE_NO_MATCHING_-SELECT_ITEM	SQLE_NO_MATCHING_SELECT_ITEM(-812)
SQLE_NO_PRIMARY_KEY	SQLE_NO_PRIMARY_KEY(-118)

Member	Description
SQLE_NOERROR	SQLE_NOERROR(0)
SQLE_NOT_IMPLEMENTED	SQLE_NOT_IMPLEMENTED(-134)
SQLE_NOT_SUPPORTED_-IN_ULTRALITE	SQLE_NOT_SUPPORTED_IN_ULTRALITE(-749)
SQLE_NOTFOUND	SQLE_NOTFOUND(100)
SQLE_OVERFLOW_ERROR	SQLE_OVERFLOW_ERROR(-158)
SQLE_PAGE_SIZE_INVALID	SQLE_PAGE_SIZE_INVALID(-644)
SQLE_PERMISSION_DENIED	SQLE_PERMISSION_DENIED(-121)
SQLE_PRIMARY_KEY_NOT_-UNIQUE	SQLE_PRIMARY_KEY_NOT_UNIQUE(-193)
SQLE_PRIMARY_KEY_-TWICE	SQLE_PRIMARY_KEY_TWICE(-126)
SQLE_PRIMARY_KEY_-VALUE_REF	SQLE_PRIMARY_KEY_VALUE_REF(-198)
SQLE_PUBLICATION_NOT_-FOUND	SQLE_PUBLICATION_NOT_FOUND(-280)
SQLE_RESOURCE_-GOVERNOR_EXCEEDED	SQLE_RESOURCE_GOVERNOR_EXCEEDED(-685)
SQLE_ROW_DELETED_TO_-MAINTAIN_REFERENTIAL_-INTEGRITY	SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_-INTEGRITY(137)
SQLE_ROW_DROPPED_-DURING_SCHEMA_-UPGRADE	SQLE_ROW_DROPPED_DURING_SCHEMA_UPGRADE(130)
SQLE_SCHEMA_UPGRADE_-NOT_ALLOWED	SQLE_SCHEMA_UPGRADE_NOT_ALLOWED(-953)
SQLE_SERVER_-SYNCHRONIZATION_ERROR	SQLE_SERVER_SYNCHRONIZATION_ERROR(-857)
SQLE_START_STOP_-DATABASE_DENIED	SQLE_START_STOP_DATABASE_DENIED(-75)
SQLE_STRING_RIGHT_-TRUNCATION	SQLE_STRING_RIGHT_TRUNCATION(-638)

Member	Description
SQLE_SUBQUERY_SELECT_-LIST	SQLE_SUBQUERY_SELECT_LIST(-151)
SQLE_SYNC_INFO_INVALID	SQLE_SYNC_INFO_INVALID(-956)
SQLE_SYNC_STATUS_-UNKNOWN	SQLE_SYNC_STATUS_UNKNOWN(-952)
SQLE_SYNCHRONIZATION_-NOT_FOUND	SQLE_SYNCHRONIZATION_NOT_FOUND(-767)
SQLE_SYNTAX_ERROR	SQLE_SYNTAX_ERROR(-131)
SQLE_TABLE_HAS_-PUBLICATIONS	SQLE_TABLE_HAS_PUBLICATIONS(-281)
SQLE_TABLE_IN_USE	SQLE_TABLE_IN_USE(-214)
SQLE_TABLE_NOT_FOUND	SQLE_TABLE_NOT_FOUND(-141)
SQLE_TOO_MANY_-CONNECTIONS	SQLE_TOO_MANY_CONNECTIONS(-102)
SQLE_TOO_MANY_TEMP_-TABLES	SQLE_TOO_MANY_TEMP_TABLES(-817)
SQLE_ULTRALITE_-DATABASE_NOT_FOUND	SQLE_ULTRALITE_DATABASE_NOT_FOUND(-954)
SQLE_ULTRALITE_OBJ_-CLOSED	SQLE_ULTRALITE_OBJ_CLOSED(-908)
SQLE_ULTRALITE_-RUNTIME_LIBRARY_-MISMATCH	SQLE_ULTRALITE_RUNTIME_LIBRARY_MISMATCH(-977)
SQLE_UNABLE_TO_-CONNECT	SQLE_UNABLE_TO_CONNECT(-105)
SQLE_UNABLE_TO_START_-DATABASE	SQLE_UNABLE_TO_START_DATABASE(-82)
SQLE_UNABLE_TO_START_-DATABASE_VER_NEWER	SQLE_UNABLE_TO_START_DATABASE_VER_NEWER(-934)
SQLE_UNCOMMITTED_-TRANSACTIONS	SQLE_UNCOMMITTED_TRANSACTIONS(-755)
SQLE_UNKNOWN_FUNC	SQLE_UNKNOWN_FUNC(-148)

Member	Description
SQLE_UNKNOWN_USERID	SQLE_UNKNOWN_USERID(-140)
SQLE_UNRECOGNIZED_-OPTION	SQLE_UNRECOGNIZED_OPTION(-1002)
SQLE_UNSUPPORTED_-CHARACTER_SET_ERROR	SQLE_UNSUPPORTED_CHARACTER_SET_ERROR(-869)
SQLE_UPLOAD_FAILED_-AT_SERVER	SQLE_UPLOAD_FAILED_AT_SERVER(-794)
SQLE_WRONG_NUM_OF_-INSERT_COLS	SQLE_WRONG_NUM_OF_INSERT_COLS(-207)
SQLE_WRONG_-PARAMETER_COUNT	SQLE_WRONG_PARAMETER_COUNT(-154)

SQLException class

Represents a SQL error returned by the UltraLite database.

Prototypes

```
' Visual Basic
Public Class SQLException
  Inherits ApplicationException

// C#
public class SQLException :
  ApplicationException
```

Remarks

This class is not serializable under the .NET Compact Framework.

SQLException members

Public instance constructors

Member	Description
SQLException constructor	Creates a SQLException with the given error code.

Public instance properties

Member	Description
ErrorCode property	The SQL code returned by the database.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the System.Exception instance that caused the current exception.
Message (inherited from Exception)	Gets a message that describes the current exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public instance methods

Member	Description
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the System.Exception that is the root cause of one or more subsequent exceptions.
GetObjectData method	Populates a SerializationInfo with the data needed to serializes this SQLException.
ToString (inherited from Exception)	Creates and returns a string representation of the current exception.

Protected instance properties

Member	Description
HResult (inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

SQLException constructor

Creates a SQLException with the given error code.

Prototypes

```
' Visual Basic
Overloads Public Sub New( _
  ByVal code As SQLCode, _
  ByVal s1 As String, _
  ByVal s2 As String, _
  ByVal s3 As String _
)

// C#
public SQLException(
  SQLCode code,
  string  s1,
  string  s2,
  string  s3
);
```

Parameters

- **code** the code of the exception.
- **s1** first string for formatted message.
- **s2** second string for formatted message.
- **s3** third string for formatted message.

Remarks

The message string corresponding to the specified SQLCode enumeration is retrieved from the **iAnywhere.UltraLite.resources** assembly. Resources are

searched for, by culture, using the following order: System.Globalization.CultureInfo.CurrentUICulture, then System.Globalization.CultureInfo.CurrentCulture, and finally culture “EN”.

ErrorCode property

The SQL code returned by the database.

Prototypes

```
' Visual Basic
Public Readonly Property ErrorCode As SQLCode

// C#
public SQLCode ErrorCode {get;}
```

GetObjectData method

Populates a SerializationInfo with the data needed to serializes this SQLException.

Prototypes

```
' Visual Basic
Overrides Public Sub GetObjectData( _
  ByVal info As System.Runtime.Serialization.SerializationInfo, _
  ByVal context As System.Runtime.Serialization.StreamingContext _
) _
  Implements ISerializable.GetObjectData

// C#
public override void GetObjectData(
  System.Runtime.Serialization.SerializationInfo info,
  System.Runtime.Serialization.StreamingContext context
);
```

Parameters

- **info** the SerializationInfo to populate with data.
- **context** the destination for this serialization.

Remarks

This method is not supported under the .NET Compact Framework.

Implements

ISerializable.GetObjectData

SQLType enumeration

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

Prototypes

' Visual Basic
Public Enum **SQLType**

// C#
public enum **SQLType**

Remarks

The table below lists which .NET types are compatible with each SQL type. In the case of integral types, table columns can always be set using smaller integer types and can be set using larger types as long as the actual value is within the range of the SQL type. For example:

```
   [Visual Basic]
   Dim t as iAnywhere.UltraLite.Table
   Dim s as Short
   ...
   ' Column 1 is of type S_LONG (Equivalent to a .NET
'System.Int32')
   t.SetShort( 1, s )  ' will succeed for all values in
s
   t.SetLong( 1, 10 )  ' (long)10 is ok
   t.SetLong( 1, Int32.MaxValue + 1 )  ' throws
SQLException

               ' Column 2 is of type U_SHORT
   Dim i as Integer = t.GetInt( 2 )  ' will succeed for
all column values
   t.SetInt( 2, 0x0ffff )  ' largest valid 'int' column
can be set with
   t.SetInt( 2, -1 )  ' SQLException

               [C#]
   Table t;
   short s;
   ...
   // Column 1 is of type S_LONG (Equivalent to a .NET
'System.Int32')
   t.SetShort( 1, s );  // will succeed for all values
in s
   t.SetLong( 1, 10 );  // (long)10 is ok
   t.SetLong( 1, Int32.MaxValue + 1 );  // throws
SQLException

               // Column 2 is of type U_SHORT
   int i = t.GetInt( 2 );  // will succeed for all
column values
   t.SetInt( 2, 0x0ffff );  // largest valid 'int'
column can be set with
   t.SetInt( 2, -1 );  // SQLException
```

SQL Type	Compatible .NET Type	C# Built-in Type	Visual Basic Built-in Type
BINARY	System.Byte[], or System.Guid if size is 16	byte[]	Byte()
BIT	System.Boolean	bool	Boolean
CHAR	System.String	String	String
DATE	System.DateTime	DateTime No built-in type.	Date
DOUBLE	System.Double	double	Double

SQL Type	Compatible .NET Type	C# Built-in Type	Visual Basic Built-in Type
LONGBINARY	System.Byte[]	byte[]	Byte()
LONGVARCHAR	System.String	String	String
NUMERIC	System.String	String	String
REAL	System.Single	float	Single
S_BIG	System.Int64	long	Long
S_LONG	System.Int32	int	Integer
S_SHORT	System.Int16	short	Short
TIME	System.TimeSpan	TimeSpan No built-in type.	TimeSpan No built-in type.
TIMESTAMP	System.DateTime	DateTime No built-in type.	Date
TINY	System.Byte	byte	Byte
U_BIG	System.UInt64	ulong	UInt64 No built-in type.
U_LONG	System.UInt32	uint	UInt32 No built-in type.
U_SHORT	System.UInt16	ushort	UInt16 No built-in type.
UUID	System.Guid	Guid No built-in type.	Guid No built-in type.

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

Members

Member	Description
BAD_INDEX	Column type returned if bad column indentifier is passed to GetColumnSQLType method.
BINARY	Binary data, with a specified length.
BIT	1-bit flag.
CHAR	Character data, with a specified length.
DATE	Date information.
DOUBLE	Double precision floating point number (8 bytes).
LONGBINARY	Binary data, with variable length.

Member	Description
LONGVARCHAR	Character data, with variable length.
NUMERIC	Exact numerical data, with a specified precision and scale.
REAL	Single precision floating point number (4 bytes).
S_BIG	Signed 64-bit integer.
S_LONG	Signed 32-bit integer.
S_SHORT	Signed 16-bit integer.
TIME	Time information.
TIMESTAMP	Timestamp information (date, time).
TINY	Unsigned 8-bit integer.
U_BIG	Unsigned 64-bit integer.
U_LONG	Unsigned 32-bit integer.
U_SHORT	Unsigned 16-bit integer.
UUID	Universally Unique Identifer.

StreamErrorCode enumeration

Enumerates the error codes that may be reported by streams during synchronization.

Prototypes

' Visual Basic
Public Enum **StreamErrorCode**

// C#
public enum **StreamErrorCode**

Members

Member	Description
ACTSYNC_NO_PORT	ACTSYNC_NO_PORT(75)
ACTSYNC_NOT_INSTALLED	ACTSYNC_NOT_INSTALLED(76)
CREATE_RANDOM_OBJECT	CREATE_RANDOM_OBJECT(17)
DEQUEUING_CONNECTION	DEQUEUING_CONNECTION(19)
END_READ	END_READ(11)
END_WRITE	END_WRITE(10)
GENERATE_RANDOM	GENERATE_RANDOM(14)
HTTP_BAD_STATUS_CODE	HTTP_BAD_STATUS_CODE(86)
HTTP_BUFFER_SIZE_OUT_-OF_RANGE	HTTP_BUFFER_SIZE_OUT_OF_RANGE(79)
HTTP_CHUNK_LEN_BAD_-CHARACTER	HTTP_CHUNK_LEN_BAD_CHARACTER(85)
HTTP_CHUNK_LEN_-ENCODED_MISSING	HTTP_CHUNK_LEN_ENCODED_MISSING(84)
HTTP_CLIENT_ID_NOT_SET	HTTP_CLIENT_ID_NOT_SET(78)
HTTP_CONTENT_TYPE_-NOT_SPECIFIED	HTTP_CONTENT_TYPE_NOT_SPECIFIED(77)
HTTP_CRLF_ENCODED_-MISSING	HTTP_CRLF_ENCODED_MISSING(81)
HTTP_CRLF_MISSING	HTTP_CRLF_MISSING(82)
HTTP_EXPECTED_POST	HTTP_EXPECTED_POST(89)
HTTP_EXTRA_DATA_END_-READ	HTTP_EXTRA_DATA_END_READ(80)

Member	Description
HTTP_NO_CONTD_-CONNECTION	HTTP_NO_CONTD_CONNECTION(83)
HTTP_UNABLE_TO_PARSE_-COOKIE	HTTP_UNABLE_TO_PARSE_COOKIE(88)
HTTP_UNKNOWN_-TRANSFER_ENCODING	HTTP_UNKNOWN_TRANSFER_ENCODING(87)
HTTP_VERSION	HTTP_VERSION(54)
INIT_RANDOM	INIT_RANDOM(15)
LOAD_NETWORK_LIBRARY	LOAD_NETWORK_LIBRARY(74)
MEMORY_ALLOCATION	MEMORY_ALLOCATION(6)
NONE	NONE(0)
NOT_IMPLEMENTED	NOT_IMPLEMENTED(12)
PARAMETER	PARAMETER(1)
PARAMETER_NOT_-BOOLEAN	PARAMETER_NOT_BOOLEAN(4)
PARAMETER_NOT_HEX	PARAMETER_NOT_HEX(5)
PARAMETER_NOT_UINT32	PARAMETER_NOT_UINT32(2)
PARAMETER_NOT_UINT32_-RANGE	PARAMETER_NOT_UINT32_RANGE(3)
PARSE	PARSE(7)
READ	READ(8)
SECURE_ADD_CERTIFICATE	SECURE_ADD_CERTIFICATE(39)
SECURE_ADD_TRUSTED_-CERTIFICATE	SECURE_ADD_TRUSTED_CERTIFICATE(48)
SECURE_CERTIFICATE_-CHAIN_FUNC	SECURE_CERTIFICATE_CHAIN_FUNC(28)
SECURE_CERTIFICATE_-CHAIN_LENGTH	SECURE_CERTIFICATE_CHAIN_LENGTH(22)
SECURE_CERTIFICATE_-CHAIN_REF	SECURE_CERTIFICATE_CHAIN_REF(29)

Member	Description
SECURE_CERTIFICATE_-COMMON_NAME	SECURE_CERTIFICATE_COMMON_NAME(52)
SECURE_CERTIFICATE_-COMPANY_NAME	SECURE_CERTIFICATE_COMPANY_NAME(21)
SECURE_CERTIFICATE_-COMPANY_UNIT	SECURE_CERTIFICATE_COMPANY_UNIT(51)
SECURE_CERTIFICATE_-COUNT	SECURE_CERTIFICATE_COUNT(42)
SECURE_CERTIFICATE_-EXPIRED	SECURE_CERTIFICATE_EXPIRED(50)
SECURE_CERTIFICATE_-EXPIRY_DATE	SECURE_CERTIFICATE_EXPIRY_DATE(37)
SECURE_CERTIFICATE_-FILE_NOT_FOUND	SECURE_CERTIFICATE_FILE_NOT_FOUND(33)
SECURE_CERTIFICATE_-NOT_TRUSTED	SECURE_CERTIFICATE_NOT_TRUSTED(24)
SECURE_CERTIFICATE_REF	SECURE_CERTIFICATE_REF(23)
SECURE_CERTIFICATE_-ROOT	SECURE_CERTIFICATE_ROOT(20)
SECURE_CREATE_-CERTIFICATE	SECURE_CREATE_CERTIFICATE(43)
SECURE_CREATE_PRIVATE_-KEY_OBJECT	SECURE_CREATE_PRIVATE_KEY_OBJECT(49)
SECURE_DUPLICATE_-CONTEXT	SECURE_DUPLICATE_CONTEXT(25)
SECURE_ENABLE_NON_-BLOCKING	SECURE_ENABLE_NON_BLOCKING(30)
SECURE_EXPORT_-CERTIFICATE	SECURE_EXPORT_CERTIFICATE(38)
SECURE_HANDSHAKE	SECURE_HANDSHAKE(53)
SECURE_IMPORT_-CERTIFICATE	SECURE_IMPORT_CERTIFICATE(44)

Member	Description
SECURE_READ_-CERTIFICATE	SECURE_READ_CERTIFICATE(34)
SECURE_READ_PRIVATE_-KEY	SECURE_READ_PRIVATE_KEY(35)
SECURE_SET_CHAIN_-NUMBER	SECURE_SET_CHAIN_NUMBER(32)
SECURE_SET_CIPHER_-SUITES	SECURE_SET_CIPHER_SUITES(31)
SECURE_SET_IO	SECURE_SET_IO(26)
SECURE_SET_IO_-SEMANTICS	SECURE_SET_IO_SEMANTICS(27)
SECURE_SET_PRIVATE_KEY	SECURE_SET_PRIVATE_KEY(36)
SECURE_SET_PROTOCOL_-SIDE	SECURE_SET_PROTOCOL_SIDE(47)
SECURE_SET_RANDOM_-FUNC	SECURE_SET_RANDOM_FUNC(46)
SECURE_SET_RANDOM_REF	SECURE_SET_RANDOM_REF(45)
SECURE_SET_READ_FUNC	SECURE_SET_READ_FUNC(55)
SECURE_SET_WRITE_FUNC	SECURE_SET_WRITE_FUNC(56)
SECURE_TRUSTED_-CERTIFICATE_FILE_NOT_-FOUND	SECURE_TRUSTED_CERTIFICATE_FILE_NOT_FOUND(40)
SECURE_TRUSTED_-CERTIFICATE_READ	SECURE_TRUSTED_CERTIFICATE_READ(41)
SEED_RANDOM	SEED_RANDOM(16)
SHUTTING_DOWN	SHUTTING_DOWN(18)
SOCKET_BIND	SOCKET_BIND(62)
SOCKET_CLEANUP	SOCKET_CLEANUP(63)
SOCKET_CLOSE	SOCKET_CLOSE(64)
SOCKET_CONNECT	SOCKET_CONNECT(65)
SOCKET_CREATE_TCPIP	SOCKET_CREATE_TCPIP(60)

Member	Description
SOCKET_CREATE_UDP	SOCKET_CREATE_UDP(61)
SOCKET_GET_HOST_BY_-ADDR	SOCKET_GET_HOST_BY_ADDR(58)
SOCKET_GET_NAME	SOCKET_GET_NAME(66)
SOCKET_GET_OPTION	SOCKET_GET_OPTION(67)
SOCKET_HOST_NAME_-NOT_FOUND	SOCKET_HOST_NAME_NOT_FOUND(57)
SOCKET_LISTEN	SOCKET_LISTEN(69)
SOCKET_LOCALHOST_-NAME_NOT_FOUND	SOCKET_LOCALHOST_NAME_NOT_FOUND(59)
SOCKET_PORT_OUT_OF_-RANGE	SOCKET_PORT_OUT_OF_RANGE(73)
SOCKET_SELECT	SOCKET_SELECT(71)
SOCKET_SET_OPTION	SOCKET_SET_OPTION(68)
SOCKET_SHUTDOWN	SOCKET_SHUTDOWN(70)
SOCKET_STARTUP	SOCKET_STARTUP(72)
WOULD_BLOCK	WOULD_BLOCK(13)
WRITE	WRITE(9)

See also ♦ “StreamErrorCode property” on page 544

StreamErrorContext enumeration

Enumerates the basic network operation being performed when the stream errors occured.

Prototypes

' Visual Basic
Public Enum **StreamErrorContext**

// C#
public enum **StreamErrorContext**

Members

Member	Description
CLOSE	CLOSE(6)
CREATE	CREATE(3)
DESTROY	DESTROY(4)
END_READ	END_READ(11)
END_WRITE	END_WRITE(10)
GETVALUE	GETVALUE(14)
OPEN	OPEN(5)
READ	READ(7)
REGISTER	REGISTER(1)
SOFTSHUTDOWN	SOFTSHUTDOWN(13)
UNKNOWN	UNKNOWN(0)
UNREGISTER	UNREGISTER(2)
WRITE	WRITE(8)
WRITE_FLUSH	WRITE_FLUSH(9)
YIELD	YIELD(12)

See also

- "StreamErrorContext property" on page 544

StreamErrorID enumeration

Enumerates the network layers that may report errors during synchronization.

Prototypes

' Visual Basic
Public Enum **StreamErrorID**

// C#
public enum **StreamErrorID**

Members

Member	Description
ACTIVESYNC	ACTIVESYNC(22)
CERTICOM	CERTICOM(11)
CERTICOM_SSL	CERTICOM_SSL(13)
CERTICOM_TLS	CERTICOM_TLS(14)
DH_CAST	DH_CAST(9)
EMAIL	EMAIL(20)
FAKE	FAKE(2)
FILE	FILE(21)
HTTP	HTTP(7)
HTTPS	HTTPS(8)
JAVA_CERTICOM	JAVA_CERTICOM(12)
JAVA_RSA	JAVA_RSA(24)
NETTECH	NETTECH(5)
PALM_CONDUIT	PALM_CONDUIT(3)
PALM_SS	PALM_SS(4)
REPLAY	REPLAY(17)
RIMBB	RIMBB(6)
RSA_TLS	RSA_TLS(23)
SECURE	SECURE(10)
SERIAL	SERIAL(1)
STRM	STRM(18)

Member	Description
TCPIP	TCPIP(0)
UDP	UDP(19)
WIRELESS	WIRELESS(16)
WIRESTRM	WIRESTRM(15)

See also

- “StreamErrorID property” on page 544

StreamType enumeration

Enumerates the types of MobiLink synchronization streams to use for synchronization.

Prototypes

' Visual Basic
Public Enum **StreamType**

// C#
public enum **StreamType**

Remarks

For information on configuring specific stream types, refer to the **Synchronization Stream Parameters Reference** section of the **UltraLite Database User's Guide** online book.

Members

Member	Description
ACTIVE_SYNC	ActiveSync synchronization (Windows CE only). UltraLite applications should only use ActiveSync synchronization when notified to do so by the MobiLink provider for ActiveSync. An application can listen for such notification by implementing an ActiveSyncListener interface and calling SetActiveSyncListener method.
HTTP	Synchronize via HTTP. The HTTP stream uses TCP/IP as its underlying transport. UltraLite applications act as Web browsers and the MobiLink synchronization server acts as a Web server. UltraLite applications send POST requests to send data to the server and GET requests to read data from the server.
HTTPS	Synchronize via HTTPS (HTTP with RSA transport-layer security). Transport-layer security is a separately-licensable component and must be ordered before you can install it. To order this component, see the card in your SQL Anywhere Studio package, or see *http://www.sybase.com/detail?id=1015780*.
TCPIP	Synchronize via TCP/IP.
UNKNOWN	Unknown or no synchronization stream. The user has not set the stream type.

See also

- "Stream property" on page 498

SyncParms class

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

Prototypes

' Visual Basic
NotInheritable Public Class **SyncParms**

// C#
public sealed class **SyncParms**

Remarks

This class cannot be directly instantiated. Each connection has its own SyncParms instance, attached as its SyncParms property property.

Only one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) may be specified at a time. If more than one of these parameters is set to true, a SQLCode enumeration SQLException will be thrown by Synchronize method.

Other sources of SQLCode enumeration errors include not specifying a Stream property value or a Version property value.

See also

- "SyncParms members" on page 491
- "Connection class" on page 345
- "SyncParms property" on page 352
- "Synchronize method" on page 360

SyncParms members

Public instance properties

Member	Description
AuthenticationParms property	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
CheckpointStore property	Specifies whether the client should perform extra store checkpoints to control the growth of the database store during synchronization.
DisableConcurrency property	Specifies whether to disable or enable concurrent access to UltraLite while performing a synchronization.
DownloadOnly property	Specifies whether to disable or enable uploads when synchronizing.
KeepPartialDownload property	Specifies whether to disable or enable partial downloads when synchronizing.
NewPassword property	Specifies a new MobiLink password for the user specified with UserName.

Member	Description
Password property	The MobiLink password for the user specified by UserName.
PingOnly property	Specifies whether the client should only ping the MobiLink synchronization server instead of performing a real synchronization.
PublicationMask property	Specifies the publications to be synchronized.
ResumePartialDownload property	Specifies whether to resume or discard a previous partial download.
SendColumnNames property	Specifies whether the client should send column names to the MobiLink synchronization server during synchronization.
SendDownloadAck property	Specifies whether the client should send a download acknowledgement to the MobiLink synchronization server during synchronization.
Stream property	Specifies the MobiLink synchronization stream to use for synchronization.
StreamParms property	Specifies the parameters to configure the synchronization stream.
UploadOnly property	Specifies whether to disable or enable downloads when synchronizing.
UserName property	The user name that uniquely identifies the MobiLink client to the MobiLink synchronization server.
Version property	Specifies which synchronization script to use.

Public instance methods

Member	Description
CopyFrom method	Copies the properties of the specified SyncParms class object to this SyncParms class object.

Protected instance methods

Member	Description
Finalize method	Destructor which cleans up the associated JNI state.

AuthenticationParms property

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Prototypes

' Visual Basic
Public Property **AuthenticationParms** As String()

// C#
public string [] **AuthenticationParms** {get;set;}

Remarks

Specified as an array of strings, each containing an authentication parameter (null array entries will result in a synchronization error).

Only the first 255 strings are used and each string should be no longer than 128 characters (longer strings will be truncated when sent to MobiLink).

The default is no authentication parms.

CheckpointStore property

Specifies whether the client should perform extra store checkpoints to control the growth of the database store during synchronization.

Prototypes

' Visual Basic
Public Property **CheckpointStore** As Boolean

// C#
public bool **CheckpointStore** {get;set;}

Remarks

The checkpoint operation adds I/O operations for the application, and so slows synchronization. This option is most useful for large downloads with many updates. Devices with slow flash memory may not want to pay the performance penalty.

The default is false (only required checkpointing is done).

DisableConcurrency property

Specifies whether to disable or enable concurrent access to UltraLite while performing a synchronization.

Prototypes

' Visual Basic
Public Property **DisableConcurrency** As Boolean

// C#
public bool **DisableConcurrency** {get;set;}

Remarks

By default, other threads may perform UltraLite operations while a thread is synchronizing. When concurrent synchronization is disabled, other threads will block on UltraLite calls until the synchronization has completed.

DownloadOnly property

Specifies whether to disable or enable uploads when synchronizing.

Prototypes

' Visual Basic
Public Property **DownloadOnly** As Boolean

// C#
public bool **DownloadOnly** {get;set;}

Remarks

Only one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) may be specified at a time. If more than one of these parameters is set to true, a SQLCode enumeration SQLException will be thrown by Synchronize method.

The default is false.

See also

- "SyncParms class" on page 491
- "SyncParms members" on page 491
- "UploadOnly property" on page 499

KeepPartialDownload property

Specifies whether to disable or enable partial downloads when synchronizing.

Prototypes

' Visual Basic
Public Property **KeepPartialDownload** As Boolean

// C#
public bool **KeepPartialDownload** {get;set;}

Remarks

UltraLite.NET has the ability to restart downloads that fail because of communication errors or user aborts through the SyncProgressListener. UltraLite.NET processes the download as it is received. If a download is interrupted, then the partial download transaction will remain in the database and can be resumed during the next synchronization.

To indicate that UltraLite.NET should save partial downloads, specify `connection.SyncParms.KeepPartialDownload=true;` otherwise the download will be rolled back if an error occurs.

If a partial download was kept, then the output field `connection.SyncResult.SyncResult.PartialDownloadRetained` will be set to true when `connection.Synchronize()` exits.

If `PartialDownloadRetained` is set, then you can resume a download.

To do this, call `connection.Synchronize()` with `connection.SyncParms.SyncParms.ResumePartialDownload` set to true. You'll likely still want `KeepPartialDownload` set to true as well in case another communications error occurs. No upload is done if a download is skipped.

The download you receive during a resumed download will be as old as when the download originally began. If you need the most up to date data, then you can do another download immediately after the special resumed download completes.

When resuming a download, many of the SyncParms fields are not relevant. For example, the PublicationMask field is not used. You will receive the publications that you requested on the initial download. The only fields that need to be set are ResumePartialDownload property and UserName property. The fields `KeepPartialDownload` and DisableConcurrency property may be set if desired and will function as normal.

If you have a partial download and it is no longer needed, then you can call RollbackPartialDownload method to roll back the failed download transaction. Also if you attempt to synchronize again and do not specify ResumePartialDownload, then the partial download will be rolled back before the next synchronization begins.

For more information, refer to the **Restartable downloads** section of the **MobiLink Synchronization User's Guide**.

The default is false.

See also

- "SyncParms class" on page 491
- "SyncParms members" on page 491
- "PartialDownloadRetained property" on page 544
- "ResumePartialDownload property" on page 497
- "RollbackPartialDownload method" on page 359

NewPassword property

Specifies a new MobiLink password for the user specified with UserName.

Prototypes

' Visual Basic
Public Property **NewPassword** As String

// C#
public string **NewPassword** {get;set;}

Remarks

A new password will take effect after the next synchronization.

The default is null (password not replaced).

See also

- “SyncParms class” on page 491
- “SyncParms members” on page 491
- “UserName property” on page 499

Password property

The MobiLink password for the user specified by UserName.

Prototypes

' Visual Basic
Public Property **Password** As String

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

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 synchronization server.

The default is null (no password).

See also

- “SyncParms class” on page 491
- “SyncParms members” on page 491
- “NewPassword property” on page 495
- “UserName property” on page 499

PingOnly property

Specifies whether the client should only ping the MobiLink synchronization server instead of performing a real synchronization.

Prototypes

' Visual Basic
Public Property **PingOnly** As Boolean

// C#
public bool **PingOnly** {get;set;}

Remarks

Only one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) may be specified at a time. If more than one of these parameters is set to true, a SQLCode enumeration SQLException will be thrown by Synchronize method.

The default is false.

PublicationMask property

Specifies the publications to be synchronized.

Prototypes

' Visual Basic
Public Property **PublicationMask** As Integer

// C#
public int **PublicationMask** {get;set;}

Remarks

The default is SYNC_ALL_DB field.

See also

- "SyncParms class" on page 491
- "SyncParms members" on page 491
- "PublicationSchema class" on page 448

ResumePartialDownload property

Specifies whether to resume or discard a previous partial download.

Prototypes

' Visual Basic
Public Property **ResumePartialDownload** As Boolean

// C#
public bool **ResumePartialDownload** {get;set;}

Remarks

Only one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) may be specified at a time. If more than one of these parameters is set to true, a SQLCode enumeration SQLException will be thrown by Synchronize method.

The default is false.

See also

- "SyncParms class" on page 491
- "SyncParms members" on page 491
- "KeepPartialDownload property" on page 494
- "PartialDownloadRetained property" on page 544

SendColumnNames property

Specifies whether the client should send column names to the MobiLink synchronization server during synchronization.

Prototypes

' Visual Basic
Public Property **SendColumnNames** As Boolean

// C#
public bool **SendColumnNames** {get;set;}

Remarks

This parameter is typically used together with the -za or -ze option on the MobiLink synchronization server for automatically generating synchronization scripts.

The default is false (column names are not sent).

SendDownloadAck property

Specifies whether the client should send a download acknowledgement to the MobiLink synchronization server during synchronization.

Prototypes

' Visual Basic
Public Property **SendDownloadAck** As Boolean

// C#
public bool **SendDownloadAck** {get;set;}

Remarks

If the client does send a download acknowledgement, the MobiLink synchronization server worker thread must wait for the client to apply the download. If the client does not sent a download acknowledgement, the MobiLink synchronization server is freed up sooner for its next synchronization.

The default is false (no download acknowledgement is sent).

Stream property

Specifies the MobiLink synchronization stream to use for synchronization.

Prototypes

' Visual Basic
Public Property **Stream** As StreamType

// C#
public StreamType **Stream** {get;set;}

Remarks

Most synchronization streams require parameters to identify the MobiLink synchronization server address and control other behavior. These parameters are supplied by the StreamParms property property.

If the stream type is set to a value that is invalid for the platform, the stream type will be set to StreamType enumeration.

This parameter has no default value, and must be explicitly set.

See also

- "SyncParms class" on page 491
- "SyncParms members" on page 491
- "StreamType enumeration" on page 490
- "StreamParms property" on page 498

StreamParms property

Specifies the parameters to configure the synchronization stream.

Prototypes

' Visual Basic
Public Property **StreamParms** As String

// C#
public string **StreamParms** {get;set;}

Remarks

For information on configuring specific stream types, refer to the **Synchronization Stream Parameters Reference** section of the **UltraLite Database User's Guide** online book.

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"`).

The default is null.

See also

- "SyncParms class" on page 491
- "SyncParms members" on page 491
- "Stream property" on page 498
- "StreamType enumeration" on page 490

UploadOnly property

Specifies whether to disable or enable downloads when synchronizing.

Prototypes

' Visual Basic
Public Property **UploadOnly** As Boolean

// C#
public bool **UploadOnly** {get;set;}

Remarks

Only one synchronization command (DownloadOnly property, PingOnly property, ResumePartialDownload property, or UploadOnly property) may be specified at a time. If more than one of these parameters is set to true, a SQLCode enumeration SQLException will be thrown by Synchronize method.

The default is false.

See also

- "SyncParms class" on page 491
- "SyncParms members" on page 491
- "DownloadOnly property" on page 494

UserName property

The user name that uniquely identifies the MobiLink client to the MobiLink synchronization server.

Prototypes

' Visual Basic
Public Property **UserName** As String

```
// C#
public string  UserName {get;set;}
```

Remarks

MobiLink uses this value to determine the download content, to record the synchronization state, and to recover from interruptions during synchronization. This user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink synchronization server.

This parameter has no default value, and must be explicitly set.

See also

- "SyncParms class" on page 491
- "SyncParms members" on page 491
- "Password property" on page 496

Version property

Specifies which synchronization script to use.

Prototypes

```
' Visual Basic
Public Property Version As String
```

```
// C#
public string  Version {get;set;}
```

Remarks

Each synchronization script in the consolidated database is marked with a version string. For example, there may be two different download_cursor scripts, with each one identified by different a version string. The version string allows an UltraLite application to choose from a set of synchronization scripts.

This parameter has no default value, and must be explicitly set.

CopyFrom method

Copies the properties of the specified SyncParms class object to this SyncParms class object.

Prototypes

```
' Visual Basic
Public Sub CopyFrom( _
  ByVal src As SyncParms _
)
```

```
// C#
public void CopyFrom(
  SyncParms src
);
```

Parameters

- **src** object to copy from.

Finalize method

Destructor which cleans up the associated JNI state.

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

SyncProgressData class

Returns synchronization progress monitoring data.

Prototypes

' Visual Basic
Public Class **SyncProgressData**

// C#
public class **SyncProgressData**

See also

- "SyncProgressData members" on page 502
- "SyncProgressListener interface" on page 539

SyncProgressData members

Public instance properties

Member	Description
ErrorMessage property	Error message describing the error that occured during synchronization.
ReceivedBytes property	The number of bytes received so far. This information is updated for all states.
ReceivedDeletes property	The number of deleted rows received so far.
ReceivedInserts property	The number of inserted rows received so far.
ReceivedUpdates property	The number of updated rows received so far.
SentBytes property	The number of bytes sent so far. This information is updated for all states.
SentDeletes property	The number of deleted rows sent so far.
SentInserts property	The number of inserted rows sent so far.
SentUpdates property	The number of updated rows sent so far.
SQLCode property	SQL code for synchronization.
State property	The current synchronization state.
SyncParms property	Reference to the Connection's SyncParms object.
SyncResult property	Reference to the Connection's SyncResult object. This object is only updated with STATE_DONE and STATE_ERROR events.
TableCount property	A count of the tables sent or received (TableCount of TableTotal) so far.

Member	Description
TableIndex property	Index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).
TableTotal property	The number of tables being synchronized.

ErrorMessage property

Error message describing the error that occured during synchronization.

Prototypes

' Visual Basic
Public Readonly Property **ErrorMessage** As String

// C#
public string **ErrorMessage** {get;}

See also

- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540

ReceivedBytes property

The number of bytes received so far. This information is updated for all states.

Prototypes

' Visual Basic
Public Readonly Property **ReceivedBytes** As Long

// C#
public long **ReceivedBytes** {get;}

See also

- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540
- "SyncProgressState enumeration" on page 540

ReceivedDeletes property

The number of deleted rows received so far.

Prototypes

' Visual Basic
Public Readonly Property **ReceivedDeletes** As Integer

// C#
public int **ReceivedDeletes** {get;}

See also

- "SyncProgressData class" on page 502

- “SyncProgressData members” on page 502
- “SyncProgressState enumeration” on page 540
- “SyncProgressState enumeration” on page 540

ReceivedInserts property

The number of inserted rows received so far.

Prototypes

’ Visual Basic
Public Readonly Property **ReceivedInserts** As Integer

// C#
public int **ReceivedInserts** {get;}

See also

- “SyncProgressData class” on page 502
- “SyncProgressData members” on page 502
- “SyncProgressState enumeration” on page 540
- “SyncProgressState enumeration” on page 540

ReceivedUpdates property

The number of updated rows received so far.

Prototypes

’ Visual Basic
Public Readonly Property **ReceivedUpdates** As Integer

// C#
public int **ReceivedUpdates** {get;}

See also

- “SyncProgressData class” on page 502
- “SyncProgressData members” on page 502
- “SyncProgressState enumeration” on page 540
- “SyncProgressState enumeration” on page 540

SentBytes property

The number of bytes sent so far. This information is updated for all states.

Prototypes

’ Visual Basic
Public Readonly Property **SentBytes** As Long

// C#
public long **SentBytes** {get;}

See also

- “SyncProgressData class” on page 502
- “SyncProgressData members” on page 502
- “SyncProgressState enumeration” on page 540
- “SyncProgressState enumeration” on page 540

SentDeletes property

The number of deleted rows sent so far.

Prototypes

' Visual Basic
Public Readonly Property **SentDeletes** As Integer

// C#
public int **SentDeletes** {get;}

See also

- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540
- "SyncProgressState enumeration" on page 540

SentInserts property

The number of inserted rows sent so far.

Prototypes

' Visual Basic
Public Readonly Property **SentInserts** As Integer

// C#
public int **SentInserts** {get;}

See also

- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540
- "SyncProgressState enumeration" on page 540

SentUpdates property

The number of updated rows sent so far.

Prototypes

' Visual Basic
Public Readonly Property **SentUpdates** As Integer

// C#
public int **SentUpdates** {get;}

See also

- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540
- "SyncProgressState enumeration" on page 540

SQLCode property

SQL code for synchronization.

Prototypes

' Visual Basic
Public Readonly Property **SQLCode** As SQLCode

// C#
public SQLCode **SQLCode** {get;}

See also

- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540

State property

The current synchronization state.

Prototypes

' Visual Basic
Public Readonly Property **State** As SyncProgressState

// C#
public SyncProgressState **State** {get;}

See also

- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540

SyncParms property

Reference to the Connection's SyncParms object.

Prototypes

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

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

See also

- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540
- "SyncProgressState enumeration" on page 540
- "SyncParms property" on page 352

SyncResult property

Reference to the Connection's SyncResult object. This object is only updated with STATE_DONE and STATE_ERROR events.

Prototypes

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

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

See also
- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540
- "SyncProgressState enumeration" on page 540
- "SyncResult property" on page 352

TableCount property

A count of the tables sent or received (TableCount of TableTotal) so far.

Prototypes

' Visual Basic
Public Readonly Property **TableCount** As Integer

// C#
public int **TableCount** {get;}

Return value

count of tables sent or received.

See also
- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540
- "SyncProgressState enumeration" on page 540

TableIndex property

Index of the table currently being synchronized (tables are numbered 1 to DatabaseSchema.TableCount).

Prototypes

' Visual Basic
Public Readonly Property **TableIndex** As Integer

// C#
public int **TableIndex** {get;}

See also
- "SyncProgressData class" on page 502
- "SyncProgressData members" on page 502
- "SyncProgressState enumeration" on page 540
- "SyncProgressState enumeration" on page 540

TableTotal property

The number of tables being synchronized.

Prototypes

' Visual Basic
Public Readonly Property **TableTotal** As Integer

```
// C#
public int TableTotal {get;}
```

See also

- “SyncProgressData class” on page 502
- “SyncProgressData members” on page 502
- “SyncProgressState enumeration” on page 540
- “SyncProgressState enumeration” on page 540

SyncProgressDialog class

Displays a synchronization progress monitoring dialog.

Prototypes

```
' Visual Basic
Public Class SyncProgressDialog
  Inherits Form, SyncProgressListener
```

```
// C#
public class SyncProgressDialog :
  Form,
  SyncProgressListener
```

Remarks

This dialog must be used as a modal, single use dialog. When the application calls the dialog's ShowDialog method method to display the dialog, it blocks the rest of the application until synchronization has completed or the user has clicked the Cancel button.

Note: Methods documented as being "called during synchronization" must not access any connections, schemas, or tables.

See also

- "SyncProgressDialog members" on page 509
- "Synchronize method" on page 360

SyncProgressDialog members

Public instance constructors

Member	Description
SyncProgressDialog constructor	Construct a SyncProgressDialog instance to synchronize the UltraLite database using the specified connection.

Public instance properties

Member	Description
AcceptButton (inherited from Form)	Gets or sets the button on the form that is clicked when the user presses the ENTER key.
AccessibilityObject (inherited from Control)	Gets the System.Windows.Forms.AccessibleObject assigned to the control.
AccessibleDefaultActionDescription (inherited from Control)	Gets or sets the default action description of the control for use by accessibility client applications.
AccessibleDescription (inherited from Control)	Gets or sets the description of the control used by accessibility client applications.

Member	Description
AccessibleName (inherited from Control)	Gets or sets the name of the control used by accessibility client applications.
AccessibleRole (inherited from Control)	Gets or sets the accessible role of the control
ActiveControl (inherited from ContainerControl)	Gets or sets the active control on the container control.
ActiveMdiChild (inherited from Form)	Gets the currently active multiple document interface (MDI) child window.
AllowDrop (inherited from Control)	Gets or sets a value indicating whether the control can accept data that the user drags onto it.
AllowTransparency (inherited from Form)	
Anchor (inherited from Control)	Gets or sets which edges of the control are anchored to the edges of its container.
AutoScale (inherited from Form)	Gets or sets a value indicating whether the form adjusts its size to fit the height of the font used on the form and scales its controls.
AutoScaleBaseSize (inherited from Form)	Gets or sets the base size used for autoscaling of the form.
AutoScroll (inherited from Form)	Gets or sets a value indicating whether the form enables autoscrolling.
AutoScrollMargin (inherited from ScrollableControl)	Gets or sets the size of the auto-scroll margin.
AutoScrollMinSize (inherited from ScrollableControl)	Gets or sets the minimum size of the auto-scroll.
AutoScrollPosition (inherited from ScrollableControl)	Gets or sets the location of the auto-scroll position.
BackColor (inherited from Form)	
BackgroundImage (inherited from Control)	Gets or sets the background image displayed in the control.
BindingContext (inherited from ContainerControl)	
Bottom (inherited from Control)	Gets the distance between the bottom edge of the control and the top edge of its container’s client area.

Member	Description
Bounds (inherited from Control)	Gets or sets the size and location of the control including its nonclient elements.
CancelButton (inherited from Form)	Gets or sets the button control that is clicked when the user presses the ESC key.
CanFocus (inherited from Control)	Gets a value indicating whether the control can receive focus.
CanSelect (inherited from Control)	Gets a value indicating whether the control can be selected.
Capture (inherited from Control)	Gets or sets a value indicating whether the control has captured the mouse.
CausesValidation (inherited from Control)	Gets or sets a value indicating whether the control causes validation to be performed on any controls that require validation when it receives focus.
ClientRectangle (inherited from Control)	Gets the rectangle that represents the client area of the control.
ClientSize (inherited from Form)	Gets or sets the size of the client area of the form.
CompanyName (inherited from Control)	Gets the name of the company or creator of the application containing the control.
ContainsFocus (inherited from Control)	Gets a value indicating whether the control, or one of its child controls, currently has the input focus.
ContextMenu (inherited from Control)	Gets or sets the shortcut menu associated with the control.
ControlBox (inherited from Form)	Gets or sets a value indicating whether a control box is displayed in the caption bar of the form.
Controls (inherited from Control)	Gets the collection of controls contained within the control.
Created (inherited from Control)	Gets a value indicating whether the control has been created.
Cursor (inherited from Control)	Gets or sets the cursor that is displayed when the mouse pointer is over the control.
DataBindings (inherited from Control)	Gets the data bindings for the control.
DesktopBounds (inherited from Form)	Gets or sets the size and location of the form on the Windows desktop.

Member	Description
DesktopLocation (inherited from Form)	Gets or sets the location of the form on the Windows desktop.
DialogResult (inherited from Form)	Gets or sets the dialog result for the form.
DisplayRectangle (inherited from ScrollableControl)	
Disposing (inherited from Control)	Gets a value indicating whether the control is in the process of being disposed of.
Dock (inherited from Control)	Gets or sets which edge of the parent container a control is docked to.
DockPadding (inherited from ScrollableControl)	Gets the dock padding settings for all edges of the control.
Enabled (inherited from Control)	Gets or sets a value indicating whether the control can respond to user interaction.
Focused (inherited from Control)	Gets a value indicating whether the control has input focus.
Font (inherited from Control)	Gets or sets the font of the text displayed by the control.
ForeColor (inherited from Control)	Gets or sets the foreground color of the control.
FormBorderStyle (inherited from Form)	Gets or sets the border style of the form.
Handle (inherited from Control)	Gets the window handle that the control is bound to.
HasChildren (inherited from Control)	Gets a value indicating whether the control contains one or more child controls.
Height (inherited from Control)	Gets or sets the height of the control.
HelpButton (inherited from Form)	Gets or sets a value indicating whether a Help button should be displayed in the caption box of the form.
Icon (inherited from Form)	Gets or sets the icon for the form.
ImeMode (inherited from Control)	Gets or sets the Input Method Editor (IME) mode of the control.
InvokeRequired (inherited from Control)	Gets a value indicating whether the caller must call an invoke method when making method calls to the control because the caller is on a different thread than the one the control was created on.
IsAccessible (inherited from Control)	Gets or sets a value indicating whether the control is visible to accessibility applications.

Member	Description
IsDisposed (inherited from Control)	Gets a value indicating whether the control has been disposed of.
IsHandleCreated (inherited from Control)	Gets a value indicating whether the control has a handle associated with it.
IsMdiChild (inherited from Form)	Gets a value indicating whether the form is a multiple document interface (MDI) child form.
IsMdiContainer (inherited from Form)	Gets or sets a value indicating whether the form is a container for multiple document interface (MDI) child forms.
IsRestrictedWindow (inherited from Form)	
KeyPreview (inherited from Form)	Gets or sets a value indicating whether the form will receive key events before the event is passed to the control that has focus.
Left (inherited from Control)	Gets or sets the x-coordinate of a control's left edge in pixels.
Location (inherited from Control)	Gets or sets the coordinates of the upper-left corner of the control relative to the upper-left corner of its container.
MaximizeBox (inherited from Form)	Gets or sets a value indicating whether the maximize button is displayed in the caption bar of the form.
MaximumSize (inherited from Form)	Gets the maximum size the form can be resized to.
MdiChildren (inherited from Form)	Gets an array of forms that represent the multiple document interface (MDI) child forms that are parented to this form.
MdiParent (inherited from Form)	Gets or sets the current multiple document interface (MDI) parent form of this form.
Menu (inherited from Form)	Gets or sets the System.Windows.Forms.MainMenu that is displayed in the form.
MergedMenu (inherited from Form)	Gets the merged menu for the form.
MinimizeBox (inherited from Form)	Gets or sets a value indicating whether the minimize button is displayed in the caption bar of the form.
MinimumSize (inherited from Form)	Gets or sets the minimum size the form can be resized to.
Modal (inherited from Form)	Gets a value indicating whether this form is displayed modally.
Name (inherited from Control)	Gets or sets the name of the control.

Member	Description
Opacity (inherited from Form)	Gets or sets the opacity level of the form.
OwnedForms (inherited from Form)	Gets an array of System.Windows.Forms.Form objects that represent all forms that are owned by this form.
Owner (inherited from Form)	Gets or sets the form that owns this form.
Parent (inherited from Control)	Gets or sets the parent container of the control.
ParentForm (inherited from ContainerControl)	Gets the form that the container control is assigned to.
ProductName (inherited from Control)	Gets the product name of the assembly containing the control.
ProductVersion (inherited from Control)	Gets the version of the assembly containing the control.
RecreatingHandle (inherited from Control)	Gets a value indicating whether the control is currently re-creating its handle.
Region (inherited from Control)	Gets or sets the window region associated with the control.
Right (inherited from Control)	Gets the distance between the right edge of the control and the left edge of its container.
RightToLeft (inherited from Control)	Gets or sets a value indicating whether control's elements are aligned to support locales using right-to-left fonts.
ShowInTaskbar (inherited from Form)	Gets or sets a value indicating whether the form is displayed in the Windows taskbar.
Site (inherited from Control)	Gets or sets the site of the control.
Size (inherited from Form)	Gets or sets the size of the form.
SizeGripStyle (inherited from Form)	Gets or sets the style of the size grip to display in the lower-right corner of the form.
SQLCode property	The SQL error code from the last synchronization.
StartPosition (inherited from Form)	Gets or sets the starting position of the form at run time.
TabIndex (inherited from Form)	
TabStop (inherited from Control)	Gets or sets a value indicating whether the user can give the focus to this control using the TAB key.
Tag (inherited from Control)	Gets or sets the object that contains data about the control.

Member	Description
Text (inherited from Control)	Gets or sets the text associated with this control.
Top (inherited from Control)	Gets or sets the y-coordinate of the control's top edge in pixels.
TopLevel (inherited from Form)	Gets or sets a value indicating whether to display the form as a top-level window.
TopLevelControl (inherited from Control)	Gets the parent control that is not parented by another Windows Forms control. Typically, this is the outermost System.Windows.Forms.Form that the control is contained in.
TopMost (inherited from Form)	Gets or sets a value indicating whether the form should be displayed as the top-most form of your application.
TransparencyKey (inherited from Form)	Gets or sets the color that will represent transparent areas of the form.
Visible (inherited from Control)	Gets or sets a value indicating whether the control is displayed.
Width (inherited from Control)	Gets or sets the width of the control.
WindowState (inherited from Form)	Gets or sets the form's window state.
WindowTarget (inherited from Control)	

Public instance methods

Member	Description
Activate (inherited from Form)	Activates the form and gives it focus.
AddOwnedForm (inherited from Form)	Adds an owned form to this form.
BeginInvoke (inherited from Control)	Executes the specified delegate asynchronously with the specified arguments, on the thread that the control's underlying handle was created on.
BeginInvoke (inherited from Control)	Executes the specified delegate asynchronously on the thread that the control's underlying handle was created on.
BringToFront (inherited from Control)	Brings the control to the front of the z-order.
Close (inherited from Form)	Closes the form.

Member	Description
Contains (inherited from Control)	Retrieves a value indicating whether the specified control is a child of the control.
CreateControl (inherited from Control)	Forces the creation of the control, including the creation of the handle and any child controls.
CreateGraphics (inherited from Control)	Creates the System.Drawing.Graphics object for the control.
DoDragDrop (inherited from Control)	Begins a drag-and-drop operation.
EndInvoke (inherited from Control)	Retrieves the return value of the asynchronous operation represented by the System.IAsyncResult object passed.
FindForm (inherited from Control)	Retrieves the form that the control is on.
Focus (inherited from Control)	Sets input focus to the control.
GetChildAtPoint (inherited from Control)	Retrieves the child control that is located at the specified coordinates.
GetContainerControl (inherited from Control)	Returns the next System.Windows.Forms.ContainerControl up the control's chain of parent controls.
GetNextControl (inherited from Control)	Retrieves the next control forward or back in the tab order of child controls.
Hide (inherited from Control)	Conceals the control from the user.
Invalidate (inherited from Control)	Invalidates the specified region of the control (adds it to the control's update region, which is the area that will be repainted at the next paint operation), and causes a paint message to be sent to the control.
Invalidate (inherited from Control)	Invalidates the specified region of the control (adds it to the control's update region, which is the area that will be repainted at the next paint operation), and causes a paint message to be sent to the control. Optionally, invalidates the child controls assigned to the control.
Invalidate (inherited from Control)	Invalidates a specific region of the control and causes a paint message to be sent to the control.
Invalidate (inherited from Control)	Invalidates a specific region of the control and causes a paint message to be sent to the control. Optionally, invalidates the child controls assigned to the control.

Member	Description
Invalidate (inherited from Control)	Invalidates the specified region of the control (adds it to the control's update region, which is the area that will be repainted at the next paint operation), and causes a paint message to be sent to the control.
Invalidate (inherited from Control)	Invalidates the specified region of the control (adds it to the control's update region, which is the area that will be repainted at the next paint operation), and causes a paint message to be sent to the control. Optionally, invalidates the child controls assigned to the control.
Invoke (inherited from Control)	Executes the specified delegate, on the thread that owns the control's underlying window handle, with the specified list of arguments.
Invoke (inherited from Control)	Executes the specified delegate on the thread that owns the control's underlying window handle.
LayoutMdi (inherited from Form)	Arranges the multiple document interface (MDI) child forms within the MDI parent form.
PerformLayout (inherited from Control)	Forces the control to apply layout logic to all its child controls.
PerformLayout (inherited from Control)	Forces the control to apply layout logic to all its child controls.
PointToClient (inherited from Control)	Computes the location of the specified screen point into client coordinates.
PointToScreen (inherited from Control)	Computes the location of the specified client point into screen coordinates.
PreProcessMessage (inherited from Control)	Preprocesses input messages within the message loop before they are dispatched.
RectangleToClient (inherited from Control)	Computes the size and location of the specified screen rectangle in client coordinates.
RectangleToScreen (inherited from Control)	Computes the size and location of the specified client rectangle in screen coordinates.
Refresh (inherited from Control)	Forces the control to invalidate its client area and immediately redraw itself and any child controls.
RemoveOwnedForm (inherited from Form)	Removes an owned form from this form.
ResetBackColor (inherited from Control)	Resets the System.Windows.Forms.Control.BackColor property to its default value.

Member	Description
ResetBindings (inherited from Control)	Resets the System.Windows.Forms.Control.DataBindings property to its default value.
ResetCursor (inherited from Control)	Resets the System.Windows.Forms.Control.Cursor property to its default value.
ResetFont (inherited from Control)	Resets the System.Windows.Forms.Control.Fontproperty to its default value.
ResetForeColor (inherited from Control)	Resets the System.Windows.Forms.Control.ForeColor property to its default value.
ResetImeMode (inherited from Control)	Resets the System.Windows.Forms.Control.ImeMode property to its default value.
ResetRightToLeft (inherited from Control)	Resets the System.Windows.Forms.Control.RightToLeft property to its default value.
ResetText (inherited from Control)	Resets the System.Windows.Forms.Control.Text property to its default value.
ResumeLayout (inherited from Control)	Resumes normal layout logic.
ResumeLayout (inherited from Control)	Resumes normal layout logic. Optionally forces an immediate layout of pending layout requests.
Scale (inherited from Control)	Scales the control and any child controls to the specified ratio.
Scale (inherited from Control)	Scales the control and any child controls by the specified horizontal and vertical ratios.
ScrollControlIntoView (inherited from ScrollableControl)	
Select (inherited from Control)	Activates the control.
SelectNextControl (inherited from Control)	Activates the next control.
SendToBack (inherited from Control)	Sends the control to the back of the z-order.
SetAutoScrollMargin (inherited from ScrollableControl)	Sets the size of the auto-scroll margins.
SetBounds (inherited from Control)	Sets the bounds of the control to the specified location and size.

Member	Description
SetBounds (inherited from Control)	Sets the specified bounds of the control to the specified location and size.
SetDesktopBounds (inherited from Form)	Sets the bounds of the form in desktop coordinates.
SetDesktopLocation (inherited from Form)	Sets the location of the form in desktop coordinates.
Show (inherited from Control)	Displays the control to the user.
ShowDialog method	Shows the form as a modal dialog with the specified parent and performs synchronization. The dialog will close when synchronization has completed.
ShowDialog method	Shows the form as a modal dialog with no parent window and performs synchronization. The dialog will close when synchronization has completed.
ShowDialog (inherited from Form)	Shows the form as a modal dialog with the specified owner.
SuspendLayout (inherited from Control)	Temporarily suspends the layout logic for the control.
SyncProgressed method	Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue. For most event states (SyncProgressData.getState()), getMessage is invoked to formulate the message to be displayed, and the sent and received byte counts are updated. The appropriate row counts are updated for the STATE_SENDING_DATA and STATE_RECEIVED_DATA events. If a STATE_DONE or STATE_CANCELLED event is received, the dialog will be hidden after a brief pause. If a STATE_ERROR event is received, GetErrorMessage is invoked to formulate the error message, the Cancel button is changed to a Dismiss button, and the dialog is not hidden until the user clicks on the button. NOTE: The only safe UltraLite .NET API calls are schema inquiry API calls (`DatabaseSchema, TableSchema, PublicationSchema, IndexSchema`).
ToString (inherited from Form)	
Update (inherited from Control)	Causes the control to redraw the invalidated regions within its client area.

Member	Description
Validate (inherited from ContainerControl)	Validates the last invalidated control and its ancestors up through, but not including, the current control.

Public instance events

Member	Description
Activated (inherited from Form)	Occurs when the form is activated in code or by the user.
BackColorChanged (inherited from Control)	Occurs when the value of the System.Windows.Forms.Control.BackColor property changes.
BackgroundImageChanged (inherited from Control)	Occurs when the value of the System.Windows.Forms.Control.BackgroundImage property changes.
BindingContextChanged (inherited from Control)	Occurs when the value of the System.Windows.Forms.BindingContext property changes.
CausesValidationChanged (inherited from Control)	Occurs when the value of the System.Windows.Forms.Control.CausesValidation property changes.
ChangeUICues (inherited from Control)	Occurs when the focus or keyboard user interface (UI) cues change.
Click (inherited from Control)	Occurs when the control is clicked.
Closed (inherited from Form)	Occurs when the form is closed.
Closing (inherited from Form)	Occurs when the form is closing.
ContextMenuChanged (inherited from Control)	Occurs when the value of the System.Windows.Forms.Control.ContextMenu property changes.
ControlAdded (inherited from Control)	Occurs when a new control is added to the System.Windows.Forms.Control.ControlCollection .
ControlRemoved (inherited from Control)	Occurs when a control is removed from the System.Windows.Forms.Control.ControlCollection .
CursorChanged (inherited from Control)	Occurs when the value of the System.Windows.Forms.Control.Cursor property changes.
Deactivate (inherited from Form)	Occurs when the form loses focus and is not the active form.
DockChanged (inherited from Control)	Occurs when the value of the System.Windows.Forms.Control.Dock property changes.
DoubleClick (inherited from Control)	Occurs when the control is double-clicked.

Member	Description
DragDrop (inherited from Control)	Occurs when a drag-and-drop operation is completed.
DragEnter (inherited from Control)	Occurs when an object is dragged into the control's bounds.
DragLeave (inherited from Control)	Occurs when an object is dragged out of the control's bounds.
DragOver (inherited from Control)	Occurs when an object is dragged over the control's bounds.
EnabledChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.Enabled property value has changed.
Enter (inherited from Control)	Occurs when the control is entered.
FontChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.Font property value changes.
ForeColorChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.ForeColor property value changes.
GiveFeedback (inherited from Control)	Occurs during a drag operation.
GotFocus (inherited from Control)	Occurs when the control receives focus.
HandleCreated (inherited from Control)	Occurs when a handle is created for the control.
HandleDestroyed (inherited from Control)	Occurs when the control's handle is in the process of being destroyed.
HelpRequested (inherited from Control)	Occurs when the user requests help for a control.
ImeModeChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.ImeMode property has changed.
InputLanguageChanged (inherited from Form)	Occurs after the input language of the form has changed.
InputLanguageChanging (inherited from Form)	Occurs when the user attempts to change the input language for the form.
Invalidated (inherited from Control)	Occurs when a control's display requires redrawing.

Member	Description
KeyDown (inherited from Control)	Occurs when a key is pressed while the control has focus.
KeyPress (inherited from Control)	Occurs when a key is pressed while the control has focus.
KeyUp (inherited from Control)	Occurs when a key is released while the control has focus.
Layout (inherited from Control)	Occurs when a control should reposition its child controls.
Leave (inherited from Control)	Occurs when the input focus leaves the control.
Load (inherited from Form)	Occurs before a form is displayed for the first time.
LocationChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.Location property value has changed.
LostFocus (inherited from Control)	Occurs when the control loses focus.
MaximizedBoundsChanged (inherited from Form)	Occurs when the value of the System.Windows.Forms.Form.-MaximizedBounds property has changed.
MaximumSizeChanged (inherited from Form)	Occurs when the value of the System.Windows.Forms.Form.-MaximumSize property has changed.
MdiChildActivate (inherited from Form)	Occurs when a multiple document interface (MDI) child form is activated or closed within an MDI application.
MenuComplete (inherited from Form)	Occurs when the menu of a form loses focus.
MenuStart (inherited from Form)	Occurs when the menu of a form receives focus.
MinimumSizeChanged (inherited from Form)	Occurs when the value of the System.Windows.Forms.Form.-MinimumSize property has changed.
MouseDown (inherited from Control)	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseEnter (inherited from Control)	Occurs when the mouse pointer enters the control.
MouseHover (inherited from Control)	Occurs when the mouse pointer hovers over the control.
MouseLeave (inherited from Control)	Occurs when the mouse pointer leaves the control.
MouseMove (inherited from Control)	Occurs when the mouse pointer is moved over the control.

Member	Description
MouseUp (inherited from Control)	Occurs when the mouse pointer is over the control and a mouse button is released.
MouseWheel (inherited from Control)	Occurs when the mouse wheel moves while the control has focus.
Move (inherited from Control)	Occurs when the control is moved.
Paint (inherited from Control)	Occurs when the control is redrawn.
ParentChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.Parent property value changes.
QueryAccessibilityHelp (inherited from Control)	Occurs when System.Windows.Forms.AccessibleObject is providing help to accessibility applications.
QueryContinueDrag (inherited from Control)	Occurs during a drag-and-drop operation and allows the drag source to determine whether the drag-and-drop operation should be canceled.
Resize (inherited from Control)	Occurs when the control is resized.
RightToLeftChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.RightToLeft property value changes.
SizeChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.Size property value changes.
StyleChanged (inherited from Control)	Occurs when the control style changes.
SystemColorsChanged (inherited from Control)	Occurs when the system colors change.
TabIndexChanged (inherited from Form)	
TabStopChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.TabStop property value changes.
TextChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.Text property value changes.
Validated (inherited from Control)	Occurs when the control is finished validating.
Validating (inherited from Control)	Occurs when the control is validating.
VisibleChanged (inherited from Control)	Occurs when the System.Windows.Forms.Control.Visible property value changes.

Protected instance properties

Member	Description
CreateParams (inherited from Form)	
DefaultImeMode (inherited from Form)	Gets the default Input Method Editor (IME) mode supported by the control.
DefaultSize (inherited from Form)	
FontHeight (inherited from Control)	Gets or sets the height of the font of the control.
HScroll (inherited from ScrollableControl)	Gets or sets a value indicating whether the horizontal scroll bar is visible.
MaximizedBounds (inherited from Form)	Gets and sets the size of the form when it is maximized.
RenderRightToLeft (inherited from Control)	
ResizeRedraw (inherited from Control)	Gets or sets a value indicating whether the control redraws itself when resized.
ShowFocusCues (inherited from Control)	Gets a value indicating whether the control should display focus rectangles.
ShowKeyboardCues (inherited from Control)	Gets a value indicating whether the control should display keyboard shortcuts.
VScroll (inherited from ScrollableControl)	Gets or sets a value indicating whether the vertical scroll bar is visible.

Protected instance methods

Member	Description
AccessibilityNotifyClients (inherited from Control)	Notifies the accessibility client applications of the specified System.Windows.Forms.AccessibleEvents for the specified child control.
ActivateMdiChild (inherited from Form)	
AdjustFormScrollbars (inherited from Form)	

Member	Description
ApplyAutoScaling (inherited from Form)	
CenterToParent (inherited from Form)	
CenterToScreen (inherited from Form)	
CreateAccessibilityInstance (inherited from Control)	Creates a new accessibility object for the control.
CreateControlsInstance (inherited from Form)	
CreateHandle (inherited from Form)	
DefWndProc (inherited from Form)	
DestroyHandle (inherited from Control)	Destroys the handle associated with the control.
Dispose method	Clean up any resources being used.
GetErrorMessage method	Returns a message for the STATE_ERROR event state. The message is formatted with the SQL code of the error. This method is called during synchronization.
GetMessage method	Returns a message for the event state. The message is formatted with `data.GetTableIndex(), data.GetTableCount()`, and `data.GetTableTotal()`. This method is called during synchronization.
GetScrollState (inherited from ScrollableControl)	
GetString method	Returns localized string (in languages supported by UltraLite.NET).
GetStyle (inherited from Control)	Retrieves the value of the specified control style bit for the control.
GetTopLevel (inherited from Control)	Determines if the control is a top-level control.
InitLayout (inherited from Control)	Called after the control has been added to another container.

Member	Description
InvokeGotFocus (inherited from Control)	Raises the System.Windows.Forms.Control.GotFocus event for the specified control.
InvokeLostFocus (inherited from Control)	Raises the System.Windows.Forms.Control.LostFocus event for the specified control.
InvokeOnClick (inherited from Control)	Raises the System.Windows.Forms.Control.Click event for the specified control.
InvokePaint (inherited from Control)	Raises the System.Windows.Forms.Control.Paint event for the specified control.
InvokePaintBackground (inherited from Control)	Raises the event for the specified control.
IsInputChar (inherited from Control)	Determines if a character is an input character that the control recognizes.
IsInputKey (inherited from Control)	Determines whether the specified key is a regular input key or a special key that requires preprocessing.
NotifyInvalidate (inherited from Control)	
OnActivated (inherited from Form)	Raises the System.Windows.Forms.Form.Activated event.
OnBackColorChanged (inherited from Control)	Raises the System.Windows.Forms.Control.BackColorChanged event.
OnBackgroundImageChanged (inherited from Control)	Raises the System.Windows.Forms.Control.-BackgroundImageChanged event.
OnBindingContextChanged (inherited from Control)	Raises the System.Windows.Forms.Control.-BindingContextChangedevent.
OnCausesValidationChanged (inherited from Control)	Raises the System.Windows.Forms.Control.-CausesValidationChangedevent.
OnChangeUICues (inherited from Control)	Raises the System.Windows.Forms.Control.ChangeUICues event.
OnClick (inherited from Control)	Raises the System.Windows.Forms.Control.Click event.
OnClosed (inherited from Form)	Raises the System.Windows.Forms.Form.Closed event.
OnClosing method	Overide closing behavior to cancel synchronization and not close until synchronization has processed the cancel request.

Member	Description
OnContextMenuChanged (inherited from Control)	Raises the System.Windows.Forms.Control.ContextMenuChangedevent.
OnControlAdded (inherited from Control)	Raises the System.Windows.Forms.Control.ControlAdded event.
OnControlRemoved (inherited from ContainerControl)	
OnCreateControl (inherited from Form)	
OnCursorChanged (inherited from Control)	Raises the System.Windows.Forms.Control.CursorChangedevent.
OnDeactivate (inherited from Form)	Raises the System.Windows.Forms.Form.Deactivate event.
OnDockChanged (inherited from Control)	Raises the System.Windows.Forms.Control.DockChangedevent.
OnDoubleClick (inherited from Control)	Raises the System.Windows.Forms.Control.DoubleClick event.
OnDragDrop (inherited from Control)	Raises the System.Windows.Forms.Control.DragDrop event.
OnDragEnter (inherited from Control)	Raises the System.Windows.Forms.Control.DragEnter event.
OnDragLeave (inherited from Control)	Raises the System.Windows.Forms.Control.DragLeave event.
OnDragOver (inherited from Control)	Raises the System.Windows.Forms.Control.DragOver event.
OnEnabledChanged (inherited from Control)	Raises the System.Windows.Forms.Control.EnabledChanged event.
OnEnter (inherited from Control)	Raises the System.Windows.Forms.Control.Enter event.
OnFontChanged (inherited from Form)	
OnForeColorChanged (inherited from Control)	Raises the System.Windows.Forms.Control.ForeColorChangedevent.
OnGiveFeedback (inherited from Control)	Raises the System.Windows.Forms.Control.GiveFeedback event.

Member	Description
OnGotFocus (inherited from Control)	Raises the System.Windows.Forms.Control.GotFocus event.
OnHandleCreated (inherited from Form)	
OnHandleDestroyed (inherited from Form)	
OnHelpRequested (inherited from Control)	Raises the System.Windows.Forms.Control.HelpRequested event.
OnImeModeChanged (inherited from Control)	Raises the System.Windows.Forms.Control.ImeModeChanged event.
OnInputLanguageChanged (inherited from Form)	Raises the System.Windows.Forms.Form.InputLanguageChanged event.
OnInputLanguageChanging (inherited from Form)	Raises the System.Windows.Forms.Form.InputLanguageChanging event.
OnInvalidated (inherited from Control)	Raises the System.Windows.Forms.Control.Invalidated event.
OnKeyDown (inherited from Control)	Raises the System.Windows.Forms.Control.KeyDown event.
OnKeyPress (inherited from Control)	Raises the System.Windows.Forms.Control.KeyPress event.
OnKeyUp (inherited from Control)	Raises the System.Windows.Forms.Control.KeyUp event.
OnLayout (inherited from ScrollableControl)	
OnLeave (inherited from Control)	Raises the System.Windows.Forms.Control.Leave event.
OnLoad (inherited from Form)	Raises the System.Windows.Forms.Form.Load event.
OnLocationChanged (inherited from Control)	Raises the System.Windows.Forms.Control.LocationChanged event.
OnLostFocus (inherited from Control)	Raises the System.Windows.Forms.Control.LostFocus event.
OnMaximizedBoundsChanged (inherited from Form)	Raises the System.Windows.Forms.Form.MaximizedBoundsChanged event.

Member	Description
OnMaximumSizeChanged (inherited from Form)	Raises the System.Windows.Forms.Form.MaximumSizeChanged event.
OnMdiChildActivate (inherited from Form)	Raises the System.Windows.Forms.Form.MdiChildActivate event.
OnMenuComplete (inherited from Form)	Raises the System.Windows.Forms.Form.MenuComplete event.
OnMenuStart (inherited from Form)	Raises the System.Windows.Forms.Form.MenuStart event.
OnMinimumSizeChanged (inherited from Form)	Raises the System.Windows.Forms.Form.MinimumSizeChanged event.
OnMouseDown (inherited from Control)	Raises the System.Windows.Forms.Control.MouseDown event.
OnMouseEnter (inherited from Control)	Raises the System.Windows.Forms.Control.MouseEnter event.
OnMouseHover (inherited from Control)	Raises the System.Windows.Forms.Control.MouseHover event.
OnMouseLeave (inherited from Control)	Raises the System.Windows.Forms.Control.MouseLeave event.
OnMouseMove (inherited from Control)	Raises the System.Windows.Forms.Control.MouseMove event.
OnMouseUp (inherited from Control)	Raises the System.Windows.Forms.Control.MouseUp event.
OnMouseWheel (inherited from ScrollableControl)	
OnMove (inherited from Control)	Raises the System.Windows.Forms.Control.Move event.
OnNotifyMessage (inherited from Control)	Notifies the control of Windows messages.
OnPaint (inherited from Form)	
OnPaintBackground (inherited from Control)	Paints the background of the control.
OnParentBackColorChanged (inherited from Control)	Raises the System.Windows.Forms.Control.BackColorChanged event when the System.Windows.Forms.Control.BackColor property value of the control's container changes.

Member	Description
OnParentBackgroundImageChanged (inherited from Control)	Raises the System.Windows.Forms.Control.BackgroundImageChanged event when the System.Windows.Forms.Control.BackgroundImage property value of the control's container changes.
OnParentBindingContextChanged (inherited from Control)	Raises the System.Windows.Forms.Control.BindingContextChanged event when the System.Windows.Forms.Control.BindingContext property value of the control's container changes.
OnParentChanged (inherited from Control)	Raises the System.Windows.Forms.Control.ParentChanged event.
OnParentEnabledChanged (inherited from Control)	Raises the System.Windows.Forms.Control.EnabledChanged event when the System.Windows.Forms.Control.Enabled property value of the control's container changes.
OnParentFontChanged (inherited from Control)	Raises the System.Windows.Forms.Control.FontChanged event when the System.Windows.Forms.Control.Font property value of the control's container changes.
OnParentForeColorChanged (inherited from Control)	Raises the System.Windows.Forms.Control.ForeColorChanged event when the System.Windows.Forms.Control.ForeColor property value of the control's container changes.
OnParentRightToLeftChanged (inherited from Control)	Raises the System.Windows.Forms.Control.RightToLeftChanged event when the System.Windows.Forms.Control.RightToLeft property value of the control's container changes.
OnParentVisibleChanged (inherited from Control)	Raises the System.Windows.Forms.Control.VisibleChanged event when the System.Windows.Forms.Control.Visible property value of the control's container changes.
OnQueryContinueDrag (inherited from Control)	Raises the System.Windows.Forms.Control.QueryContinueDrag event.
OnResize (inherited from Form)	
OnRightToLeftChanged (inherited from Control)	Raises the System.Windows.Forms.Control.RightToLeftChangedevent.
OnSizeChanged (inherited from Control)	Raises the System.Windows.Forms.Control.SizeChanged event.
OnStyleChanged (inherited from Form)	
OnSystemColorsChanged (inherited from Control)	Raises the System.Windows.Forms.Control.SystemColorsChanged event.

Member	Description
OnTabIndexChanged (inherited from Control)	Raises the System.Windows.Forms.Control.TabIndexChanged event.
OnTabStopChanged (inherited from Control)	Raises the System.Windows.Forms.Control.TabStopChanged event.
OnTextChanged (inherited from Form)	
OnValidated (inherited from Control)	Raises the System.Windows.Forms.Control.Validated event.
OnValidating (inherited from Control)	Raises the System.Windows.Forms.Control.Validating event.
OnVisibleChanged (inherited from Form)	
ProcessCmdKey (inherited from Form)	
ProcessDialogChar (inherited from ContainerControl)	
ProcessDialogKey (inherited from Form)	
ProcessKeyEventArgs (inherited from Control)	Processes a key message and generates the appropriate control events.
ProcessKeyPreview (inherited from Form)	
ProcessMnemonic (inherited from ContainerControl)	
ProcessTabKey (inherited from Form)	
RaiseDragEvent (inherited from Control)	
RaiseKeyEvent (inherited from Control)	
RaiseMouseEvent (inherited from Control)	
RaisePaintEvent (inherited from Control)	

Member	Description
RecreateHandle (inherited from Control)	Forces the re-creation of the handle for the control.
ResetMouseEventArgs (inherited from Control)	
ResetValues method	Initializes displayed values of this dialog.
RtlTranslateAlignment (inherited from Control)	Converts the specified System.Windows.Forms.HorizontalAlignment to the appropriate System.Windows.Forms.HorizontalAlignment to support right-to-left text.
RtlTranslateAlignment (inherited from Control)	Converts the specified System.Windows.Forms.LeftRightAlignment to the appropriate System.Windows.Forms.LeftRightAlignment to support right-to-left text.
RtlTranslateAlignment (inherited from Control)	Converts the specified System.Drawing.ContentAlignment to the appropriate System.Drawing.ContentAlignment to support right-to-left text.
RtlTranslateContent (inherited from Control)	Converts the specified System.Drawing.ContentAlignment to the appropriate System.Drawing.ContentAlignment to support right-to-left text.
RtlTranslateHorizontal (inherited from Control)	Converts the specified System.Windows.Forms.HorizontalAlignment to the appropriate System.Windows.Forms.HorizontalAlignment to support right-to-left text.
RtlTranslateLeftRight (inherited from Control)	Converts the specified System.Windows.Forms.LeftRightAlignment to the appropriate System.Windows.Forms.LeftRightAlignment to support right-to-left text.
ScaleCore (inherited from Form)	
Select (inherited from Form)	
SetBoundsCore (inherited from Form)	
SetClientSizeCore (inherited from Form)	
SetDisplayRectLocation (inherited from ScrollableControl)	
SetScrollState (inherited from ScrollableControl)	
SetStyle (inherited from Control)	Sets the specified style bit to the specified value.

Member	Description
SetTopLevel (inherited from Control)	Sets the control as the top-level control.
SetVisibleCore (inherited from Form)	
UpdateBounds (inherited from Control)	Updates the bounds of the control with the current size and location.
UpdateBounds (inherited from Control)	Updates the bounds of the control with the specified size and location.
UpdateBounds (inherited from Control)	Updates the bounds of the control with the specified size, location, and client size.
UpdateDefaultButton (inherited from Form)	
UpdateStyles (inherited from Control)	Forces the assigned styles to be reapplied to the control.
UpdateZOrder (inherited from Control)	Updates the control in its parent's z-order.
WndProc (inherited from Form)	

SyncProgressDialog constructor

Construct a SyncProgressDialog instance to synchronize the UltraLite database using the specified connection.

Prototypes

```
' Visual Basic
Public Sub New( _
  ByVal dbMgr As DatabaseManager, _
  ByVal conn As Connection _
)

// C#
public SyncProgressDialog(
  DatabaseManager dbMgr,
  Connection conn
);
```

Parameters

- **dbMgr** application's DatabaseManager.
- **conn** connection on which to synchronize.

Remarks

This dialog should be shown as a modal dialog using ShowDialog method.

SQLCode property

The SQL error code from the last synchronization.

Prototypes

```
' Visual Basic
Public Readonly Property SQLCode As SQLCode

// C#
public SQLCode SQLCode {get;}
```

Property value

SQL error code

See also

- "SyncProgressDialog class" on page 509
- "SyncProgressDialog members" on page 509
- "SQLCode property" on page 534

Dispose method

Clean up any resources being used.

Prototypes

```
' Visual Basic
Overloads Overrides Protected Sub Dispose( _
  ByVal disposing As Boolean _
)

// C#
protected override void Dispose(
  bool disposing
);
```

Parameters

- **disposing** true to release both managed and unmanaged resources; false to release only unmanaged resources.

GetErrorMessage method

Returns a message for the STATE_ERROR event state. The message is formatted with the SQL code of the error.

This method is called during synchronization.

Prototypes

```
' Visual Basic
Protected Function GetErrorMessage( _
  ByVal sqlCode As SQLCode, _
  ByVal errMsg As String, _
  ByVal uploadOK As Boolean _
) As String
```

```
// C#
protected string  GetErrorMessage(
  SQLCode sqlCode,
  string  errMsg,
  bool uploadOK
);
```

Parameters

- **sqlCode** SQL code of the error.
- **errMsg** error message associated with the SQL code.
- **uploadOK** status of upload.

Return value

message for event.

See also

- "SyncProgressDialog class" on page 509
- "SyncProgressDialog members" on page 509
- "SQLCode property" on page 505
- "SyncResult property" on page 506

GetMessage method

Returns a message for the event state. The message is formatted with `data.GetTableIndex(), data.GetTableCount()`, and `data.GetTableTotal()`.

This method is called during synchronization.

Prototypes

```
' Visual Basic
Protected Function GetMessage( _
  ByVal progress As SyncProgressData _
) As String
```

```
// C#
protected string  GetMessage(
  SyncProgressData progress
);
```

Parameters

- **progress** synchronization progress data.

Return value

message for event.

GetString method

Returns localized string (in languages supported by UltraLite.NET).

Prototypes

```
' Visual Basic
Protected Function GetString( _
  ByVal name As String _
) As String
```

```
// C#
protected string  GetString(
  string  name
);
```

Parameters

- **name** name of string.

Return value

localized string.

OnClosing method

Overide closing behavior to cancel synchronization and not close until synchronization has processed the cancel request.

Prototypes

```
' Visual Basic
Overrides Protected Sub OnClosing( _
  ByVal e As System.ComponentModel.CancelEventArgs _
)
```

```
// C#
protected override void OnClosing(
  System.ComponentModel.CancelEventArgs e
);
```

Parameters

- **e** a `CancelEventArgs` that contains the event data.

ResetValues method

Initializes displayed values of this dialog.

Prototypes

```
' Visual Basic
Protected Sub ResetValues()
```

```
// C#
protected void ResetValues();
```

ShowDialog method

Shows the form as a modal dialog with the specified parent and performs synchronization. The dialog will close when synchronization has completed.

Prototypes

```
' Visual Basic
Overloads Public Function ShowDialog( _
  ByVal parent As System.Windows.Forms.Form _
) As System.Windows.Forms.DialogResult
```

```
// C#
public System.Windows.Forms.DialogResult ShowDialog(
  System.Windows.Forms.Form parent
);
```

Parameters

- **parent** any System.Windows.Forms.Form object that represents the top-level window that will own the modal dialog.

Return value

System.Windows.Forms.DialogResult.OK if synchronization completed successfully, System.Windows.Forms.DialogResult.Cancel if synchronization was cancelled by the user, and System.Windows.Forms.DialogResult.Abort if an error occurred during synchronization (see SQLCode property for more specific error reporting).

Remarks

This method is recommended over ShowDialog().

The parent will be disabled while this dialog is active.

ShowDialog method

Shows the form as a modal dialog with no parent window and performs synchronization. The dialog will close when synchronization has completed.

Prototypes

' Visual Basic
Overloads Public Function **ShowDialog()** As System.Windows.Forms.DialogResult

// C#
public System.Windows.Forms.DialogResult **ShowDialog();**

Return value

System.Windows.Forms.DialogResult.OK if synchronization completed successfully, System.Windows.Forms.DialogResult.Cancel if synchronization was cancelled by the user, and System.Windows.Forms.DialogResult.Abort if an error occurred during synchronization (see SQLCode property for more specific error reporting).

Remarks

ShowDialog method is recommended over this method.

SyncProgressed method

Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue.

For most event states (SyncProgressData.getState()), getMessage is invoked to formulate the message to be displayed, and the sent and received byte counts are updated. The appropriate row counts are updated for the STATE_SENDING_DATA and STATE_RECEIVED_DATA events.

If a STATE_DONE or STATE_CANCELLED event is received, the dialog will be hidden after a brief pause.

If a STATE_ERROR event is received, GetErrorMessage is invoked to formulate the error message, the Cancel button is changed to a Dismiss button, and the dialog is not hidden until the user clicks on the button.

NOTE: The only safe UltraLite .NET API calls are schema inquiry API calls (`DatabaseSchema, TableSchema, PublicationSchema, IndexSchema`).

Prototypes

```
' Visual Basic
Public Function SyncProgressed( _
  ByVal data As SyncProgressData _
) As Boolean _
  Implements SyncProgressListener.SyncProgressed

// C#
public bool SyncProgressed(
  SyncProgressData data
);
```

Parameters

- **data** object containing latest synchronization progress data.

Return value

true to cancel synchronization, or false to continue.

Implements

"SyncProgressListener interface" on page 539

SyncProgressListener interface

The listener interface for receiving synchronization progress events.

Prototypes

```
' Visual Basic
Public Interface SyncProgressListener
```

```
// C#
public interface SyncProgressListener
```

See also

- "SyncProgressListener members" on page 539
- "Synchronize method" on page 360

SyncProgressListener members

Public instance methods

Member	Description
SyncProgressed method	Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue.

SyncProgressed method

Invoked during synchronization to inform the user of progress. This method should return true to cancel synchronization or return false to continue.

Prototypes

```
' Visual Basic
Public Function SyncProgressed( _
  ByVal data As SyncProgressData _
) As Boolean
```

```
// C#
public bool SyncProgressed(
  SyncProgressData data
);
```

Parameters

- **data** object containing the latest synchronization progress data.

Return value

This method should return true to cancel synchronization or return false to continue.

Remarks

No UltraLite.NET API methods should be invoked during a `SyncProgressed` call.

SyncProgressState enumeration

Enumerates all the states that can occur while synchronizing.

Prototypes

' Visual Basic
Public Enum **SyncProgressState**

// C#
public enum **SyncProgressState**

Members

Member	Description
STATE_CANCELLED	Synchronization has been cancelled.
STATE_COMMITTING_-DOWNLOAD	The download is being committed. The final count of rows received is included with this event. See ReceivedBytes property, ReceivedInserts property, ReceivedUpdates property, and ReceivedDeletes property.
STATE_CONNECTING	The synchronization stream has been built, but not yet opened.
STATE_DISCONNECTING	The synchronization stream is about to be closed.
STATE_DONE	Synchronization has successfully completed. The connection's SyncResult property object has been updated.
STATE_ERROR	Synchronization has completed, but an error occurred. Check SyncResult property, ErrorMessage property, and SQLCode property for details.
STATE_FINISHING_UPLOAD	The upload is completing. The final count of rows sent is included with this event. See SentBytes property, SentInserts property, SentUpdates property, and SentDeletes property.
STATE_LAST	Last state entered by synchronization. This state will always be entered and always after any other state. Other SyncProgressData fields may not contain valid data.
STATE_RECEIVING_DATA	Data for the current table is being received. ReceivedBytes property, ReceivedInserts property, ReceivedUpdates property, and ReceivedDeletes property have been updated.
STATE_RECEIVING_TABLE	A table is being received. Progress can be monitored using TableIndex property and TableCount property.
STATE_RECEIVING_-UPLOAD_ACK	An acknowledgement that the upload is complete is being received.

Member	Description
STATE_ROLLING_BACK_-DOWNLOAD	Synchronization is rolling back the download because an error was encountered during the download. The error will be reported with a subsequent STATE_ERROR progress report.
STATE_SENDING_DATA	Data for the current table is being sent. SentBytes property, SentInserts property, SentUpdates property, and SentDeletes property have been updated.
STATE_SENDING_-DOWNLOAD_ACK	An acknowledgement that the download is complete is being sent.
STATE_SENDING_HEADER	The synchronization stream has been opened and the header is about to be sent.
STATE_SENDING_TABLE	A table is being sent. Progress can be monitored using TableIndex property and TableCount property.
STATE_STARTING	No synchronization actions have been taken yet.

See also

- "SyncProgressData class" on page 502

SyncResult class

Represents the status of the last synchronization.

Prototypes

```
' Visual Basic
Public Class SyncResult
```

```
// C#
public class SyncResult
```

Remarks

This class cannot be directly instantiated. Each connection has its own SyncResult instance, attached as its SyncResult property property. A SyncResult instance is only valid while that connection is open.

See also

- "SyncResult members" on page 542
- "SyncResult property" on page 352
- "Synchronize method" on page 360

SyncResult members

Public instance properties

Member	Description
AuthStatus property	The authorization status code for the last synchronization attempt. See AuthStatusCode enumeration for possible values.
AuthValue property	The return value from custom user authentication synchronization scripts.
IgnoredRows property	True if any uploaded rows were ignored during the last synchronization, false if no rows were ignored.
PartialDownloadRetained property	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.
StreamErrorCode property	Returns the error reported by the stream itself. See StreamErrorCode property for a list of known codes.
StreamErrorContext property	The basic network operation being performed when the stream error occurred. See StreamErrorContext property for a list of known contexts.
StreamErrorID property	ID of network layer reporting an error. See StreamErrorID property for a list of known IDs.
StreamErrorSystem property	The stream error system-specific code.
Timestamp property	The timestamp of the last synchronization.

Member	Description
UploadOK property	True if the last upload synchronization was successful, false if the last upload synchronization was unsuccessful.

Protected instance methods

Member	Description
Finalize method	Destructor which cleans up the associated JNI state.

AuthStatus property

The authorization status code for the last synchronization attempt. See AuthStatusCode enumeration for possible values.

Prototypes

' Visual Basic
Public Readonly Property **AuthStatus** As AuthStatusCode

// C#
public AuthStatusCode **AuthStatus** {get;}

AuthValue property

The return value from custom user authentication synchronization scripts.

Prototypes

' Visual Basic
Public Readonly Property **AuthValue** As Long

// C#
public long **AuthValue** {get;}

IgnoredRows property

True if any uploaded rows were ignored during the last synchronization, false if no rows were ignored.

Prototypes

' Visual Basic
Public Readonly Property **IgnoredRows** As Boolean

// C#
public bool **IgnoredRows** {get;}

See also

- "SyncResult class" on page 542
- "SyncResult members" on page 542
- "DownloadOnly property" on page 494

PartialDownloadRetained property

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.

Prototypes

' Visual Basic
Public Readonly Property **PartialDownloadRetained** As Boolean

// C#
public bool **PartialDownloadRetained** {get;}

See also

- "SyncResult class" on page 542
- "SyncResult members" on page 542
- "KeepPartialDownload property" on page 494

StreamErrorCode property

Returns the error reported by the stream itself. See StreamErrorCode property for a list of known codes.

Prototypes

' Visual Basic
Public Readonly Property **StreamErrorCode** As StreamErrorCode

// C#
public StreamErrorCode **StreamErrorCode** {get;}

StreamErrorContext property

The basic network operation being performed when the stream error occurred. See StreamErrorContext property for a list of known contexts.

Prototypes

' Visual Basic
Public Readonly Property **StreamErrorContext** As StreamErrorContext

// C#
public StreamErrorContext **StreamErrorContext** {get;}

StreamErrorID property

ID of network layer reporting an error. See StreamErrorID property for a list of known IDs.

Prototypes

' Visual Basic
Public Readonly Property **StreamErrorID** As StreamErrorID

// C#
public StreamErrorID **StreamErrorID** {get;}

StreamErrorSystem property

The stream error system-specific code.

Prototypes

' Visual Basic
Public Readonly Property **StreamErrorSystem** As Integer

// C#
public int **StreamErrorSystem** {get;}

Timestamp property

The timestamp of the last synchronization.

Prototypes

' Visual Basic
Public Readonly Property **Timestamp** As Date

// C#
public DateTime **Timestamp** {get;}

UploadOK property

True if the last upload synchronization was successful, false if the last upload synchronization was unsuccessful.

Prototypes

' Visual Basic
Public Readonly Property **UploadOK** As Boolean

// C#
public bool **UploadOK** {get;}

Finalize method

Destructor which cleans up the associated JNI state.

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

Table class

Represents a table in an UltraLite database.

Prototypes

```
' Visual Basic
Public Class Table
  Inherits Cursor
```

```
// C#
public class Table :
  Cursor
```

Remarks

This class cannot be directly instantiated. Tables are created using the GetTable method method of the Connection class class.

Table members

Public instance properties

Member	Description
IsBOF property (inherited from Cursor)	Returns true if the current row position is before the first row.
IsEOF property (inherited from Cursor)	Returns true if the current row position is after the last row.
IsOpen property (inherited from Cursor)	Checks whether this cursor is currently open.
RowCount property (inherited from Cursor)	Returns the number of rows in the cursor.
Schema property	Holds the schema of this table. This property is only valid while its connection is open.

Public instance methods

Member	Description
AppendBytes method	Appends the specified subset of the specified array of Bytes to the new value for the specified SQLType enumeration column.
AppendChars method	Appends the specified subset of the specified array of System.Chars to the new value for the specified SQLType enumeration column.
Close method (inherited from Cursor)	Closes the cursor.
Delete method	Deletes the current row.

Member	Description
DeleteAllRows method	Deletes all rows in the table.
Fill method (inherited from Cursor)	Fills a ADO .NET DataTable object with the rows in the cursor.
FindBegin method	Prepares to perform a new Find on this table.
FindFirst method	Move forwards through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.
FindFirst method	Move forwards through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.
FindLast method	Move backwards through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.
FindLast method	Move backwards through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.
FindNext method	Continues a FindFirst method search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.
FindNext method	Continues a FindFirst method search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or partial set of values in the current index.
FindPrevious method	Continues a FindLast method search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.
FindPrevious method	Continues a FindLast method search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or partial set of values in the current index.
GetBoolean method (inherited from Cursor)	Returns the value for the specified column as a Boolean.
GetBytes method (inherited from Cursor)	Returns the value for the specified column as an array of Bytes. Only valid for columns of type SQLType enumeration, SQLType enumeration, or SQLType enumeration.
GetBytes method (inherited from Cursor)	Copies a subset of the value for the specified SQLType enumeration column, beginning at the specified offset, to the specified offset of the destination Byte array.

Member	Description
GetChars method (inherited from Cursor)	Copies a subset of the value for the specified SQLType enumeration column, beginning at the specified offset, to the specified offset of the destination System.Char array.
GetDouble method (inherited from Cursor)	Returns the value for the specified column as a Double.
GetFloat method (inherited from Cursor)	Returns the value for the specified column as a Single.
GetInt method (inherited from Cursor)	Returns the value for the specified column as an Int32.
GetLong method (inherited from Cursor)	Returns the value for the specified column as an Int64.
GetShort method (inherited from Cursor)	Returns the value for the specified column as an Int16.
GetString method (inherited from Cursor)	Returns the value for the specified column as a String.
GetTime method (inherited from Cursor)	Returns the value for the specified column as a TimeSpan with millisecond accuracy.
GetTimestamp method (inherited from Cursor)	Returns the value for the specified column as a DateTime with millisecond accuracy.
GetUInt method (inherited from Cursor)	Returns the value for the specified column as a UInt32.
GetULong method (inherited from Cursor)	Returns the value for the specified column as a UInt64.
GetUShort method (inherited from Cursor)	Returns the value for the specified column as a UInt16.
GetUUID method (inherited from Cursor)	Returns the value for the specified column as a UUID (Guid).
Insert method	Inserts a new row with the current column values (specified using the set methods). Each insert must be preceded by a call to InsertBegin method.
InsertBegin method	Prepares to insert a new row into this table by setting all current column values to their default values.
IsNull method (inherited from Cursor)	Checks whether the value from the specified column is NULL.

Member	Description
LookupBackward method	Move backwards through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.
LookupBackward method	Move backwards through the table from the beginning, looking for a row that matches or is less than a value or partial set of values in the current index.
LookupBegin method	Prepares to perform a new lookup on this table. The value(s) to search for are specified by calling the appropriate set*Type* method(s) on the columns in the index this table was opened with.
LookupForward method	Move forwards through the table from the beginning, looking for a row that matches or is greater than a value or full set of values in the current index.
LookupForward method	Move forwards through the table from the beginning, looking for a row that matches or is greater than a value or partial set of values in the current index.
MoveAfterLast method (inherited from Cursor)	Position to after the last row of the cursor.
MoveBeforeFirst method (inherited from Cursor)	Position to before the first row of the cursor.
MoveFirst method (inherited from Cursor)	Position to the first row of the cursor.
MoveLast method (inherited from Cursor)	Position to the last row of the cursor.
MoveNext method (inherited from Cursor)	Position to the next row or after last.
MovePrevious method (inherited from Cursor)	Position to the previous row or before first.
MoveRelative method (inherited from Cursor)	Position relative to the current row.
Open method	Opens this table for data access using its primary key.
Open method	Opens this table for data access using the specified index.
SetBoolean method	Sets the value for the specified column using a Boolean.
SetBytes method	Sets the value for the specified column using an array of Bytes.

Member	Description
SetDouble method	Sets the value for the specified column using a Double.
SetFloat method	Sets the value for the specified column using a Single.
SetInt method	Sets the value for the specified column using an Int32.
SetLong method	Sets the value for the specified column using an Int64.
SetNull method	Sets a column to `NULL`.
SetShort method	Sets the value for the specified column using an Int16.
SetString method	Sets the value for the specified column using a String.
SetTime method	Sets the value for the specified column using a TimeSpan.
SetTimestamp method	Sets the value for the specified column using a DateTime.
SetToDefault method	Sets the value for the specified column to its default value.
SetUInt method	Sets the value for the specified column using an UInt32.
SetULong method	Sets the value for the specified column using a UInt64.
SetUShort method	Sets the value for the specified column using a UInt16.
SetUUID method	Sets the value for the specified column using a Guid.
Truncate method	Deletes all rows in the table while temporarily activating a stop synchronization delete.
Update method	Updates the current row with the current column values (specified using the set methods). Each update must be preceded by a call to UpdateBegin method.
UpdateBegin method	Prepares to update the current row in this table.

Protected instance methods

Member	Description
Finalize method	Finalize method

Schema property

Holds the schema of this table. This property is only valid while its connection is open.

Prototypes

```
' Visual Basic
Public Readonly Property Schema As TableSchema
```

```
// C#
public TableSchema Schema {get;}
```

See also

- "Table class" on page 546
- "Table members" on page 546
- "TableSchema class" on page 585

AppendBytes method

Appends the specified subset of the specified array of Bytes to the new value for the specified SQLType enumeration column.

Prototypes

```
' Visual Basic
Public Sub AppendBytes( _
  ByVal columnID As Short, _
  ByVal val As Byte(), _
  ByVal srcOffset As Integer, _
  ByVal count As Integer _
)
```

```
// C#
public void AppendBytes(
  short columnID,
  byte[] val,
  int srcOffset,
  int count
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the value to append to the current new value for the column.
- **srcOffset** start position in the source array.
- **count** the number of bytes to be copied.

Remarks

The bytes at position *srcOffset* (starting from 0) through `srcOffset+count-1` of the array *val* are appended to the value for the specified column.

When inserting, InsertBegin method initializes the new value to the column's default value. The data in the row is not actually changed until you execute an Insert method, and that change is not permanent until it is committed.

When updating, the first append on a column will clear the current value prior to appending the new value.

If any of the following are true, a SQLException class with code SQLCode enumeration is thrown and the destination is not modified:

For other errors, a SQLException class with the appropriate error code is thrown.

- *val* is null.
- *srcOffset* is negative.
- *count* is negative.
- `srcOffset+count` is greater than `val.Length`.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetColumnID method” on page 395
- “SetToDefault method” on page 577
- “SetNull method” on page 572
- “AppendChars method” on page 552
- “SetBytes method” on page 566
- “SetBoolean method” on page 565
- “SetDouble method” on page 567
- “SetFloat method” on page 568
- “SetInt method” on page 569
- “SetLong method” on page 571
- “SetShort method” on page 573
- “SetString method” on page 574
- “SetTime method” on page 575
- “SetTimestamp method” on page 576
- “SetUInt method” on page 578
- “SetULong method” on page 579
- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584
- “Schema property” on page 550
- “GetColumnSQLType method” on page 397

AppendChars method

Appends the specified subset of the specified array of System.Chars to the new value for the specified SQLType enumeration column.

Prototypes

```
' Visual Basic
Public Sub AppendChars( _
  ByVal columnID As Short, _
  ByVal val As Char(), _
  ByVal srcOffset As Integer, _
  ByVal count As Integer _
)

// C#
public void AppendChars(
  short columnID,
  char[] val,
  int srcOffset,
  int count
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** The value to append to the current new value for the column.
- **srcOffset** start position in the source array.
- **count** the number of bytes to be copied.

Remarks

The characters at position *srcOffset* (starting from 0) through `srcOffset+count-1` of the array *val* are appended to the value for the specified column. When inserting, InsertBegin method initializes the new value to the column's default value. The data in the row is not actually changed until you execute an Insert method, and that change is not permanent until it is committed.

When updating, the first append on a column will clear the current value prior to appending the new value.

If any of the following is true, a SQLException class with code SQLCode enumeration is thrown and the destination is not modified:

For other errors, a SQLException class with the appropriate error code is thrown.

- *val* is null.
- *srcOffset* is negative.
- *count* is negative.
- `srcOffset+count` is greater than `value.Length`.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546

- “GetColumnID method” on page 395
- “SetToDefault method” on page 577
- “SetNull method” on page 572
- “AppendBytes method” on page 551
- “SetBytes method” on page 566
- “SetBoolean method” on page 565
- “SetDouble method” on page 567
- “SetFloat method” on page 568
- “SetInt method” on page 569
- “SetLong method” on page 571
- “SetShort method” on page 573
- “SetString method” on page 574
- “SetTime method” on page 575
- “SetTimestamp method” on page 576
- “SetUInt method” on page 578
- “SetULong method” on page 579
- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584
- “Schema property” on page 550
- “GetColumnSQLType method” on page 397

Delete method

Deletes the current row.

Prototypes

' Visual Basic
Public Sub **Delete()**

// C#
public void **Delete();**

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “StartSynchronizationDelete method” on page 359
- “StopSynchronizationDelete method” on page 360
- “AutoCommit field” on page 347

DeleteAllRows method

Deletes all rows in the table.

Prototypes

' Visual Basic
Public Sub **DeleteAllRows()**

// C#
public void **DeleteAllRows();**

Remarks

In some applications, it can be useful to delete all rows from a table before downloading a new set of data into the table. Rows can be deleted from the UltraLite database without being deleted from the consolidated database using StopSynchronizationDelete method.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "Truncate method" on page 583
- "AutoCommit field" on page 347

Finalize method

Finalize method

Prototypes

' Visual Basic
Overrides Protected Sub **Finalize()**

// C#
protected override void **Finalize();**

FindBegin method

Prepares to perform a new Find on this table.

Prototypes

' Visual Basic
Public Sub **FindBegin()**

// C#
public void **FindBegin();**

Remarks

The value(s) to search for are specified by calling the appropriate set*Type* method(s) on the columns in the index this table was opened with.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "FindFirst method" on page 556
- "FindFirst method" on page 556
- "FindLast method" on page 557
- "FindLast method" on page 557

FindFirst method

Move forwards through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindFirst() As Boolean
```

```
// C#
public bool FindFirst();
```

Return value

true if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure the cursor position is after the last row (IsEOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "FindBegin method" on page 555
- "FindNext method" on page 558
- "FindPrevious method" on page 559

FindFirst method

Move forwards through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindFirst( _
  ByVal numColumns As Short _
) As Boolean
```

```
// C#
public bool FindFirst(
  short numColumns
);
```

Parameters

- **numColumns** for composite indexes, the number of columns to use in the find. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a *numColumns* value of 1.

Return value

true if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure the cursor position is after the last row (IsEOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “FindBegin method” on page 555
- “FindNext method” on page 559
- “FindPrevious method” on page 560

FindLast method

Move backwards through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindLast() As Boolean

// C#
public bool FindLast();
```

Return value

true if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure the cursor position is before the first row (IsBOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “FindBegin method” on page 555
- “FindNext method” on page 558
- “FindPrevious method” on page 559

FindLast method

Move backwards through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindLast( _
  ByVal numColumns As Short _
) As Boolean
```

```
// C#
public bool FindLast(
  short numColumns
);
```

Parameters

- **numColumns** for composite indexes, the number of columns to use in the find. For example, if you have a three column index, and you want to find a value that matches based on the first column only, you should set the value for the first column, then supply a *numColumns* value of 1.

Return value

true if successful, false otherwise

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure the cursor position is before the first row (IsBOF property).

Each search must be preceded by a call to FindBegin method.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "FindBegin method" on page 555
- "FindNext method" on page 559
- "FindPrevious method" on page 560

FindNext method

Continues a FindFirst method search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindNext() As Boolean

// C#
public bool FindNext();
```

Return value

true if successful, false otherwise.

Remarks

The cursor is left on the next row if it exactly matches the index value. On failure the cursor position is after the last row (IsEOF property).

`FindNext` behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546

- "FindFirst method" on page 556

FindNext method

Continues a FindFirst method search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindNext( _
  ByVal numColumns As Short _
) As Boolean
```

```
// C#
public bool FindNext(
  short numColumns
);
```

Parameters

- **numColumns** for composite indexes, the number of columns to use in the find. For example, if you have a three column index, and you want to find a value that matches based on the first column only, you should set the value for the first column, and then supply a *numColumns* value of 1.

Return value

true if successful, false otherwise.

Remarks

The cursor is left on the next row if it exactly matches the index value. On failure the cursor position is after the last row (IsEOF property).

`FindNext` behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "FindFirst method" on page 556

FindPrevious method

Continues a FindLast method search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindPrevious() As Boolean
```

```
// C#
public bool FindPrevious();
```

Return value

true if successful, false otherwise.

Remarks

The cursor is left on the previous row if it exactly matches the index value. On failure the cursor position is before the first row (IsBOF property).

`FindPrevious` behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "FindLast method" on page 557

FindPrevious method

Continues a FindLast method search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function FindPrevious( _
  ByVal numColumns As Short _
) As Boolean
```

```
// C#
public bool FindPrevious(
  short numColumns
);
```

Parameters

- **numColumns** for composite indexes, the number of columns to use in the find. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, then supply a *numColumns* value of 1.

Return value

true if successful, false otherwise.

Remarks

The cursor is left on the previous row if it exactly matches the index value. On failure the cursor position is before the first row (IsBOF property).

`FindPrevious` behavior is undefined if the column values being searched for are modified during a row update.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "FindLast method" on page 557

Insert method

Inserts a new row with the current column values (specified using the set methods).

Each insert must be preceded by a call to InsertBegin method.

Prototypes

' Visual Basic
Public Sub **Insert()**

// C#
public void **Insert();**

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "AutoCommit field" on page 347

InsertBegin method

Prepares to insert a new row into this table by setting all current column values to their default values.

Prototypes

' Visual Basic
Public Sub **InsertBegin()**

// C#
public void **InsertBegin();**

Remarks

Call the appropriate Set*Type* or Append*Type* 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 permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "Insert method" on page 560

LookupBackward method

Move backwards through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.

Prototypes

' Visual Basic
Overloads Public Function **LookupBackward()** As Boolean

// C#
public bool **LookupBackward();**

Return value

true if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is less than the index value. On failure (no rows less than the value being looked for) the cursor position is before the first row (IsBOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "LookupBegin method" on page 563

LookupBackward method

Move backwards through the table from the beginning, looking for a row that matches or is less than a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function LookupBackward( _
  ByVal numColumns As Short _
) As Boolean
```

```
// C#
public bool LookupBackward(
  short numColumns
);
```

Parameters

- **numColumns** for composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a *numColumns* value of 1.

Return value

true if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is less than the index value. On failure (no rows less than the value being looked for) the cursor position is before the first row (IsBOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "LookupBegin method" on page 563

LookupBegin method

Prepares to perform a new lookup on this table. The value(s) to search for are specified by calling the appropriate set*Type* method(s) on the columns in the index this table was opened with.

Prototypes

' Visual Basic
Public Sub **LookupBegin()**

// C#
public void **LookupBegin();**

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "LookupForward method" on page 563
- "LookupForward method" on page 564
- "LookupBackward method" on page 561
- "LookupBackward method" on page 562

LookupForward method

Move forwards through the table from the beginning, looking for a row that matches or is greater than a value or full set of values in the current index.

Prototypes

' Visual Basic
Overloads Public Function **LookupForward()** As Boolean

// C#
public bool **LookupForward();**

Return value

true if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (no rows greater than the value being looked for) the cursor position is after the last row (IsEOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "LookupBegin method" on page 563

LookupForward method

Move forwards through the table from the beginning, looking for a row that matches or is greater than a value or partial set of values in the current index.

Prototypes

```
' Visual Basic
Overloads Public Function LookupForward( _
  ByVal numColumns As Short _
) As Boolean

// C#
public bool LookupForward(
  short numColumns
);
```

Parameters

- **numColumns** for composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a *numColumns* value of `1`.

Return value

true if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (no rows greater than the value being looked for) the cursor position is after the last row (IsEOF property).

Each search must be preceded by a call to LookupBegin method.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "LookupBegin method" on page 563

Open method

Opens this table for data access using its primary key.

Prototypes

```
' Visual Basic
Overloads Public Sub Open()

// C#
public void Open();
```

Exceptions

- SQLException class - A SQL error occurred.

Open method

Opens this table for data access using the specified index.

Prototypes

```
' Visual Basic
Overloads Public Sub Open( _
  ByVal index As String _
)

// C#
public void Open(
  string  index
);
```

Parameters

- **index** name of index to open the table with. If `null`, the primary key is used.

Exceptions

- SQLException class - A SQL error occurred.

SetBoolean method

Sets the value for the specified column using a Boolean.

Prototypes

```
' Visual Basic
Public Sub SetBoolean( _
  ByVal columnID As Short, _
  ByVal val As Boolean _
)

// C#
public void SetBoolean(
  short columnID,
  bool val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetColumnID method” on page 395
- “SetToDefault method” on page 577

- “SetNull method” on page 572
- “AppendBytes method” on page 551
- “AppendChars method” on page 552
- “SetBytes method” on page 566
- “SetDouble method” on page 567
- “SetFloat method” on page 568
- “SetInt method” on page 569
- “SetLong method” on page 571
- “SetShort method” on page 573
- “SetString method” on page 574
- “SetTime method” on page 575
- “SetTimestamp method” on page 576
- “SetUInt method” on page 578
- “SetULong method” on page 579
- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584
- “Schema property” on page 550
- “GetColumnSQLType method” on page 397

SetBytes method

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

Prototypes

```
' Visual Basic
Public Sub SetBytes( _
  ByVal columnID As Short, _
  ByVal val As Byte() _
)

// C#
public void SetBytes(
  short columnID,
  byte[] val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

Only suitable for columns of type SQLType enumeration or SQLType enumeration, or for columns of type SQLType enumeration when val is of length 16. The data in the row is not actually changed until you execute an

Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "GetColumnID method" on page 395
- "SetToDefault method" on page 577
- "SetNull method" on page 572
- "AppendBytes method" on page 551
- "AppendChars method" on page 552
- "SetBoolean method" on page 565
- "SetDouble method" on page 567
- "SetFloat method" on page 568
- "SetInt method" on page 569
- "SetLong method" on page 571
- "SetShort method" on page 573
- "SetString method" on page 574
- "SetTime method" on page 575
- "SetTimestamp method" on page 576
- "SetUInt method" on page 578
- "SetULong method" on page 579
- "SetUShort method" on page 581
- "SetUUID method" on page 582
- "FindBegin method" on page 555
- "LookupBegin method" on page 563
- "InsertBegin method" on page 561
- "UpdateBegin method" on page 584
- "Schema property" on page 550
- "GetColumnSQLType method" on page 397

SetDouble method

Sets the value for the specified column using a Double.

Prototypes

```
' Visual Basic
Public Sub SetDouble( _
  ByVal columnID As Short, _
  ByVal val As Double _
)

// C#
public void SetDouble(
  short columnID,
  double val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetColumnID method” on page 395
- “SetToDefault method” on page 577
- “SetNull method” on page 572
- “AppendBytes method” on page 551
- “AppendChars method” on page 552
- “SetBytes method” on page 566
- “SetBoolean method” on page 565
- “SetFloat method” on page 568
- “SetInt method” on page 569
- “SetLong method” on page 571
- “SetShort method” on page 573
- “SetString method” on page 574
- “SetTime method” on page 575
- “SetTimestamp method” on page 576
- “SetUInt method” on page 578
- “SetULong method” on page 579
- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584
- “Schema property” on page 550
- “GetColumnSQLType method” on page 397

SetFloat method

Sets the value for the specified column using a Single.

Prototypes

```
' Visual Basic
Public Sub SetFloat( _
  ByVal columnID As Short, _
  ByVal val As Single _
)
```

```
// C#
public void SetFloat(
  short columnID,
  float  val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetColumnID method” on page 395
- “SetToDefault method” on page 577
- “SetNull method” on page 572
- “AppendBytes method” on page 551
- “AppendChars method” on page 552
- “SetBytes method” on page 566
- “SetBoolean method” on page 565
- “SetDouble method” on page 567
- “SetInt method” on page 569
- “SetLong method” on page 571
- “SetShort method” on page 573
- “SetString method” on page 574
- “SetTime method” on page 575
- “SetTimestamp method” on page 576
- “SetUInt method” on page 578
- “SetULong method” on page 579
- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584
- “Schema property” on page 550
- “GetColumnSQLType method” on page 397

SetInt method

Sets the value for the specified column using an Int32.

Prototypes

```
' Visual Basic
Public Sub SetInt( _
  ByVal columnID As Short, _
  ByVal val As Integer _
)

// C#
public void SetInt(
  short columnID,
  int val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetColumnID method” on page 395
- “SetToDefault method” on page 577
- “SetNull method” on page 572
- “AppendBytes method” on page 551
- “AppendChars method” on page 552
- “SetBytes method” on page 566
- “SetBoolean method” on page 565
- “SetDouble method” on page 567
- “SetFloat method” on page 568
- “SetLong method” on page 571
- “SetShort method” on page 573
- “SetString method” on page 574
- “SetTime method” on page 575
- “SetTimestamp method” on page 576
- “SetUInt method” on page 578
- “SetULong method” on page 579
- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584

- "Schema property" on page 550
- "GetColumnSQLType method" on page 397

SetLong method

Sets the value for the specified column using an Int64.

Prototypes

```
' Visual Basic
Public Sub SetLong( _
  ByVal columnID As Short, _
  ByVal val As Long _
)

// C#
public void SetLong(
  short columnID,
  long val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "GetColumnID method" on page 395
- "SetToDefault method" on page 577
- "SetNull method" on page 572
- "AppendBytes method" on page 551
- "AppendChars method" on page 552
- "SetBytes method" on page 566
- "SetBoolean method" on page 565
- "SetDouble method" on page 567
- "SetFloat method" on page 568
- "SetInt method" on page 569
- "SetShort method" on page 573
- "SetString method" on page 574
- "SetTime method" on page 575
- "SetTimestamp method" on page 576
- "SetUInt method" on page 578
- "SetULong method" on page 579

- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584
- “Schema property” on page 550
- “GetColumnSQLType method” on page 397

SetNull method

Sets a column to `NULL`.

Prototypes

```
' Visual Basic
Public Sub SetNull( _
  ByVal columnID As Short _
)

// C#
public void SetNull(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.

Remarks

The data is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetColumnID method” on page 395
- “SetToDefault method” on page 577
- “SetBoolean method” on page 565
- “SetDouble method” on page 567
- “SetFloat method” on page 568
- “SetInt method” on page 569
- “SetLong method” on page 571
- “SetShort method” on page 573
- “SetString method” on page 574
- “SetTime method” on page 575
- “SetTimestamp method” on page 576
- “SetUInt method” on page 578
- “SetULong method” on page 579
- “SetUShort method” on page 581

- "SetUUID method" on page 582
- "FindBegin method" on page 555
- "LookupBegin method" on page 563
- "InsertBegin method" on page 561
- "UpdateBegin method" on page 584
- "Schema property" on page 550
- "IsColumnNullable method" on page 593

SetShort method

Sets the value for the specified column using an Int16.

Prototypes

```
' Visual Basic
Public Sub SetShort( _
  ByVal columnID As Short, _
  ByVal val As Short _
)

// C#
public void SetShort(
  short columnID,
  short val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "GetColumnID method" on page 395
- "SetToDefault method" on page 577
- "SetNull method" on page 572
- "AppendBytes method" on page 551
- "AppendChars method" on page 552
- "SetBytes method" on page 566
- "SetBoolean method" on page 565
- "SetDouble method" on page 567
- "SetFloat method" on page 568
- "SetInt method" on page 569
- "SetLong method" on page 571

- “SetString method” on page 574
- “SetTime method” on page 575
- “SetTimestamp method” on page 576
- “SetUInt method” on page 578
- “SetULong method” on page 579
- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584
- “Schema property” on page 550
- “GetColumnSQLType method” on page 397

SetString method

Sets the value for the specified column using a String.

Prototypes

```
' Visual Basic
Public Sub SetString( _
  ByVal columnID As Short, _
  ByVal val As String _
)

// C#
public void SetString(
  short columnID,
  string  val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetColumnID method” on page 395
- “SetToDefault method” on page 577
- “SetNull method” on page 572
- “AppendBytes method” on page 551
- “AppendChars method” on page 552

- "SetBytes method" on page 566
- "SetBoolean method" on page 565
- "SetDouble method" on page 567
- "SetFloat method" on page 568
- "SetInt method" on page 569
- "SetLong method" on page 571
- "SetShort method" on page 573
- "SetTime method" on page 575
- "SetTimestamp method" on page 576
- "SetUInt method" on page 578
- "SetULong method" on page 579
- "SetUShort method" on page 581
- "SetUUID method" on page 582
- "FindBegin method" on page 555
- "LookupBegin method" on page 563
- "InsertBegin method" on page 561
- "UpdateBegin method" on page 584
- "Schema property" on page 550
- "GetColumnSQLType method" on page 397

SetTime method

Sets the value for the specified column using a TimeSpan.

Prototypes

```
' Visual Basic
Public Sub SetTime( _
  ByVal columnID As Short, _
  ByVal val As TimeSpan _
)

// C#
public void SetTime(
  short columnID,
  TimeSpan val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The set value is accurate to the millisecond and is normalized to a nonnegative value between 0 and 24 hours. The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetColumnID method” on page 395
- “SetToDefault method” on page 577
- “SetNull method” on page 572
- “AppendBytes method” on page 551
- “AppendChars method” on page 552
- “SetBytes method” on page 566
- “SetBoolean method” on page 565
- “SetDouble method” on page 567
- “SetFloat method” on page 568
- “SetInt method” on page 569
- “SetLong method” on page 571
- “SetShort method” on page 573
- “SetString method” on page 574
- “SetTime method” on page 575
- “SetUInt method” on page 578
- “SetULong method” on page 579
- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584
- “Schema property” on page 550
- “GetColumnSQLType method” on page 397

SetTimestamp method

Sets the value for the specified column using a DateTime.

Prototypes

```
' Visual Basic
Public Sub SetTimestamp( _
  ByVal columnID As Short, _
  ByVal val As Date _
)

// C#
public void SetTimestamp(
  short columnID,
  DateTime val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.

- **val** the new value for the column.

Remarks

The set value is accurate to the millisecond. The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "GetColumnID method" on page 395
- "SetToDefault method" on page 577
- "SetNull method" on page 572
- "AppendBytes method" on page 551
- "AppendChars method" on page 552
- "SetBytes method" on page 566
- "SetBoolean method" on page 565
- "SetDouble method" on page 567
- "SetFloat method" on page 568
- "SetInt method" on page 569
- "SetLong method" on page 571
- "SetShort method" on page 573
- "SetString method" on page 574
- "SetTime method" on page 575
- "SetUInt method" on page 578
- "SetULong method" on page 579
- "SetUShort method" on page 581
- "SetUUID method" on page 582
- "FindBegin method" on page 555
- "LookupBegin method" on page 563
- "InsertBegin method" on page 561
- "UpdateBegin method" on page 584
- "Schema property" on page 550
- "GetColumnSQLType method" on page 397

SetToDefault method

Sets the value for the specified column to its default value.

Prototypes

```
' Visual Basic
Public Sub SetToDefault( _
  ByVal columnID As Short _
)

// C#
public void SetToDefault(
  short columnID
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetColumnID method” on page 395
- “SetNull method” on page 572
- “SetBytes method” on page 566
- “SetBoolean method” on page 565
- “SetDouble method” on page 567
- “SetFloat method” on page 568
- “SetInt method” on page 569
- “SetLong method” on page 571
- “SetShort method” on page 573
- “SetTime method” on page 575
- “SetTimestamp method” on page 576
- “SetUInt method” on page 578
- “SetULong method” on page 579
- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584
- “Schema property” on page 550
- “GetColumnSQLType method” on page 397
- “GetColumnDefaultValue method” on page 588

SetUInt method

Sets the value for the specified column using an UInt32.

Prototypes

```
' Visual Basic
Public Sub SetUInt( _
  ByVal columnID As Short, _
  ByVal val As UInt32 _
)
```

```
// C#
public void SetUInt(
  short columnID,
  uint val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "GetColumnID method" on page 395
- "SetToDefault method" on page 577
- "SetNull method" on page 572
- "AppendBytes method" on page 551
- "AppendChars method" on page 552
- "SetBytes method" on page 566
- "SetBoolean method" on page 565
- "SetDouble method" on page 567
- "SetFloat method" on page 568
- "SetInt method" on page 569
- "SetLong method" on page 571
- "SetShort method" on page 573
- "SetString method" on page 574
- "SetTime method" on page 575
- "SetTimestamp method" on page 576
- "SetULong method" on page 579
- "SetUShort method" on page 581
- "SetUUID method" on page 582
- "FindBegin method" on page 555
- "LookupBegin method" on page 563
- "InsertBegin method" on page 561
- "UpdateBegin method" on page 584
- "Schema property" on page 550
- "GetColumnSQLType method" on page 397

SetULong method

Sets the value for the specified column using a UInt64.

Prototypes

```
' Visual Basic
Public Sub SetULong( _
  ByVal columnID As Short, _
  ByVal val As UInt64 _
)

// C#
public void SetULong(
  short columnID,
  ulong val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetColumnID method” on page 395
- “SetToDefault method” on page 577
- “SetNull method” on page 572
- “AppendBytes method” on page 551
- “AppendChars method” on page 552
- “SetBytes method” on page 566
- “SetBoolean method” on page 565
- “SetDouble method” on page 567
- “SetFloat method” on page 568
- “SetInt method” on page 569
- “SetLong method” on page 571
- “SetShort method” on page 573
- “SetString method” on page 574
- “SetTime method” on page 575
- “SetTimestamp method” on page 576
- “SetUInt method” on page 578
- “SetUShort method” on page 581
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584

- "Schema property" on page 550
- "GetColumnSQLType method" on page 397

SetUShort method

Sets the value for the specified column using a UInt16.

Prototypes

```
' Visual Basic
Public Sub SetUShort( _
  ByVal columnID As Short, _
  ByVal val As UInt16 _
)

// C#
public void SetUShort(
  short columnID,
  ushort val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "GetColumnID method" on page 395
- "SetToDefault method" on page 577
- "SetNull method" on page 572
- "AppendBytes method" on page 551
- "AppendChars method" on page 552
- "SetBytes method" on page 566
- "SetBoolean method" on page 565
- "SetDouble method" on page 567
- "SetFloat method" on page 568
- "SetInt method" on page 569
- "SetLong method" on page 571
- "SetShort method" on page 573
- "SetString method" on page 574
- "SetTime method" on page 575
- "SetTimestamp method" on page 576
- "SetUInt method" on page 578

- “SetULong method” on page 579
- “SetUUID method” on page 582
- “FindBegin method” on page 555
- “LookupBegin method” on page 563
- “InsertBegin method” on page 561
- “UpdateBegin method” on page 584
- “Schema property” on page 550
- “GetColumnSQLType method” on page 397

SetUUID method

Sets the value for the specified column using a Guid.

Prototypes

```
' Visual Basic
Public Sub SetUUID( _
  ByVal columnID As Short, _
  ByVal val As Guid _
)

// C#
public void SetUUID(
  short columnID,
  Guid val
);
```

Parameters

- **columnID** the ID number of the column. The first column in the table has an ID value of one.
- **val** the new value for the column.

Remarks

The data in the row is not actually changed until you execute an Insert method or Update method, and that change is not permanent until it is committed. Only valid for columns of type SQLType enumeration or for columns of type SQLType enumeration with length 16.

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “GetNewUUID method” on page 356
- “GetColumnID method” on page 395
- “SetToDefault method” on page 577
- “SetNull method” on page 572
- “AppendBytes method” on page 551
- “AppendChars method” on page 552
- “SetBytes method” on page 566
- “SetBoolean method” on page 565
- “SetDouble method” on page 567

- "SetFloat method" on page 568
- "SetInt method" on page 569
- "SetLong method" on page 571
- "SetShort method" on page 573
- "SetTime method" on page 575
- "SetTimestamp method" on page 576
- "SetUInt method" on page 578
- "SetULong method" on page 579
- "SetUShort method" on page 581
- "FindBegin method" on page 555
- "LookupBegin method" on page 563
- "InsertBegin method" on page 561
- "UpdateBegin method" on page 584
- "Schema property" on page 550
- "GetColumnSQLType method" on page 397
- "GetColumnSize method" on page 397

Truncate method

Deletes all rows in the table while temporarily activating a stop synchronization delete.

Prototypes

' Visual Basic
Public Sub **Truncate()**

// C#
public void **Truncate();**

Exceptions

- SQLException class - A SQL error occurred.

See also

- "Table class" on page 546
- "Table members" on page 546
- "DeleteAllRows method" on page 554
- "AutoCommit field" on page 347

Update method

Updates the current row with the current column values (specified using the set methods).

Each update must be preceded by a call to UpdateBegin method.

Prototypes

' Visual Basic
Public Sub **Update()**

// C#
public void **Update();**

Exceptions

- SQLException class - A SQL error occurred.

See also

- “Table class” on page 546
- “Table members” on page 546
- “AutoCommit field” on page 347

UpdateBegin method

Prepares to update the current row in this table.

Prototypes

```
' Visual Basic
Public Sub UpdateBegin()

// C#
public void UpdateBegin();
```

Remarks

Column values are modified by calling the appropriate set*Type* or Append*Type* method(s). The first append on a column will clear the current column value prior to appending the new value.

The data in the row is not actually changed until you call Update method, and that change is not permanent until it is committed.

Modifying columns in the index used to open the table will affect any active searches in unpredictable ways. Columns in the primary key of the table can not be updated.

Exceptions

- SQLException class - A SQL error occurred.

TableSchema class

Represents the schema of an UltraLite table.

Prototypes

' Visual Basic

```
NotInheritable Public Class TableSchema
  Inherits CursorSchema
```

// C#

```
public sealed class TableSchema :
  CursorSchema
```

Remarks

This class cannot be directly instantiated. A TableSchema class object is attached to a table as its Schema property property.

TableSchema members

Public instance properties

Member	Description
ColumnCount property (inherited from CursorSchema)	Returns the number of columns in this cursor.
IndexCount property	The number of indexes on this table.
IsNeverSynchronized property	Whether this table is marked as never being synchronized.
IsOpen property (inherited from CursorSchema)	Checks whether this cursor schema is currently open.
Name property	The name of this table.
PrimaryKey property	The index schema of the primary key for this table.
UploadsUnchangedRows property	Whether the database will upload rows which have not changed.

Public instance methods

Member	Description
GetColumnDefaultValue method	The default value of the named column.
GetColumnID method (inherited from CursorSchema)	Returns the column ID of the named column.
GetColumnName method (inherited from CursorSchema)	Returns the name of column identified by the specified column ID.

Member	Description
GetColumnPartitionSize method	Returns the global autoincrement partition size assigned to the named column.
GetColumnPrecision method (inherited from CursorSchema)	Returns the precision of the named column if the column is a numeric column (SQL type NUMERIC).
GetColumnScale method (inherited from CursorSchema)	Returns the scale of the named column if the column is a numeric column (SQL type NUMERIC).
GetColumnSize method (inherited from CursorSchema)	Returns the size of the named column if the column is a sized column (SQL type BINARY or CHAR).
GetColumnSQLType method (inherited from CursorSchema)	Gets the SQL data type of the named column.
GetIndex method	Returns the index schema of the named index.
GetIndexName method	Returns the name of the index identified by the specified index ID.
GetOptimalIndex method	The optimal index for searching a table using the named column.
IsColumnAutoIncrement method	Checks whether the named column's default is set to autoincrement.
IsColumnCurrentDate method	Checks whether the named column's default is set to the current date (SQLType.DATE).
IsColumnCurrentTime method	Checks whether the named column's default is set to the current time (SQLType.TIME).
IsColumnCurrentTimestamp method	Checks whether the named column's default is set to the current timestamp (SQLType.TIMESTAMP).
IsColumnGlobalAutoIncrement method	Checks whether the named column's default is set to global autoincrement.
IsColumnNewUUID method	Checks whether the named column's default is set to a new UUID (Guid).
IsColumnNullable method	Checks whether the named column is nullable.
IsInPublication method	Checks whether this table is contained in the named publication.

Protected instance methods

Member	Description
Finalize method	Destructor which cleans up the associated native object.

IndexCount property

The number of indexes on this table.

Prototypes

' Visual Basic
Public Readonly Property **IndexCount** As Integer

// C#
public int **IndexCount** {get;}

Remarks

Index IDs range from 1 to `IndexCount`, inclusively.

Note: Index IDs and count may change during a schema upgrade. To correctly identify an index, access it by name or refresh the cached IDs and counts after a schema upgrade.

IsNeverSynchronized property

Whether this table is marked as never being synchronized.

Prototypes

' Visual Basic
Public Readonly Property **IsNeverSynchronized** As Boolean

// C#
public bool **IsNeverSynchronized** {get;}

Property value

true if this table is marked as never being synchronized.

Remarks

Tables marked as never being synchronized are never synchronized, even if they are included in a publication. These tables are sometimes referred to as “no sync” tables.

Exceptions

- SQLException class - A SQL error occurred.

Name property

The name of this table.

Prototypes

' Visual Basic
Public Readonly Property **Name** As String

// C#
public string **Name** {get;}

Exceptions

- SQLException class - A SQL error occurred.

PrimaryKey property

The index schema of the primary key for this table.

Prototypes

```
' Visual Basic
Public Readonly Property PrimaryKey As IndexSchema

// C#
public IndexSchema PrimaryKey {get;}
```

Exceptions

- SQLException class - A SQL error occurred.

UploadsUnchangedRows property

Whether the database will upload rows which have not changed.

Prototypes

```
' Visual Basic
Public Readonly Property UploadsUnchangedRows As Boolean

// C#
public bool UploadsUnchangedRows {get;}
```

Property value

true if this table is marked to always upload all rows during synchronization.

Remarks

Tables marked as such will upload unchanged rows, as well as changed rows, when the table is synchronized. These tables are sometimes referred to as “all sync” tables.

Exceptions

- SQLException class - A SQL error occurred.

Finalize method

Destructor which cleans up the associated native object.

Prototypes

```
' Visual Basic
Overrides Protected Sub Finalize()

// C#
protected override void Finalize();
```

GetColumnDefaultValue method

The default value of the named column.

Prototypes

```
' Visual Basic
Public Function GetColumnDefaultValue( _
  ByVal name As String _
) As String

// C#
public string  GetColumnDefaultValue(
  string  name
);
```

Parameters

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

Return value — The default value of the named column as a string or null if the default value is `null`.

Exceptions
- SQLException class - A SQL error occurred.

GetColumnPartitionSize method

Returns the global autoincrement partition size assigned to the named column.

Prototypes

```
' Visual Basic
Public Function GetColumnPartitionSize( _
  ByVal name As String _
) As UInt64

// C#
public ulong GetColumnPartitionSize(
  string name
);
```

Parameters
- **name** name of the column.

Return value — Column's global autoincrement partition size.

Remarks — All global autoincrement columns in a given table share the same global autoincrement partition.

Exceptions
- SQLException class - A SQL error occurred.

See also
- "TableSchema class" on page 585
- "TableSchema members" on page 585
- "IsColumnGlobalAutoIncrement method" on page 592

GetIndex method

Returns the index schema of the named index.

Prototypes

```
' Visual Basic
Public Function GetIndex( _
  ByVal name As String _
) As IndexSchema

// C#
public IndexSchema GetIndex(
  string name
);
```

Parameters
- **name** name of the index.

Return value — index schema of the named index.

Exceptions
- SQLException class - A SQL error occurred.

GetIndexName method

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

Prototypes

```
' Visual Basic
Public Function GetIndexName( _
  ByVal indexID As Integer _
) As String

// C#
public string  GetIndexName(
  int indexID
);
```

Parameters

- **indexID** ID of the index. *indexID* must be in the range `[1,TableSchema.IndexCount]`.

Return value

name of the index.

Remarks

Index IDs and counts may change during a schema upgrade. To correctly identify an index, access it by name or refresh the cached IDs and counts after a schema upgrade.

Exceptions

- SQLException class - A SQL error occurred.

See also

- "TableSchema class" on page 585
- "TableSchema members" on page 585
- "IndexCount property" on page 587

GetOptimalIndex method

The optimal index for searching a table using the named column.

Prototypes

```
' Visual Basic
Public Function GetOptimalIndex( _
  ByVal name As String _
) As IndexSchema

// C#
public IndexSchema GetOptimalIndex(
  string  name
);
```

Parameters

- **name** name of the column.

Return value

index for the named column.

Remarks

The named column will be the first column in the index, but the index may have more than one column.

Exceptions

- SQLException class - A SQL error occurred.

IsColumnAutoIncrement method

Checks whether the named column's default is set to autoincrement.

Prototypes

```
' Visual Basic
Public Function IsColumnAutoIncrement( _
  ByVal name As String _
) As Boolean

// C#
public bool IsColumnAutoIncrement(
  string name
);
```

Parameters

- **name** name of the column.

Return value

true if the column is autoincrementing, false if it is not autoincrementing.

Exceptions

- SQLException class - A SQL error occurred.

IsColumnCurrentDate method

Checks whether the named column's default is set to the current date (SQLType.DATE).

Prototypes

```
' Visual Basic
Public Function IsColumnCurrentDate( _
  ByVal name As String _
) As Boolean

// C#
public bool IsColumnCurrentDate(
  string name
);
```

Parameters

- **name** name of the column.

Return value

True if the column defaults to the current date, false if the column does not default to the current date.

Exceptions

- SQLException class - A SQL error occurred.

IsColumnCurrentTime method

Checks whether the named column's default is set to the current time (SQLType.TIME).

Prototypes

```
' Visual Basic
Public Function IsColumnCurrentTime( _
  ByVal name As String _
) As Boolean
```

```
// C#
public bool IsColumnCurrentTime(
  string name
);
```

Parameters

- **name** name of the column.

Return value

True if the column defaults to the current time, false if the column does not default to the current time.

Exceptions

- SQLException class - A SQL error occurred.

IsColumnCurrentTimestamp method

Checks whether the named column's default is set to the current timestamp (SQLType.TIMESTAMP).

Prototypes

```
' Visual Basic
Public Function IsColumnCurrentTimestamp( _
  ByVal name As String _
) As Boolean
```

```
// C#
public bool IsColumnCurrentTimestamp(
  string name
);
```

Parameters

- **name** name of the column.

Return value

True if the column defaults to the current timestamp, false if the column does not default to the current timestamp.

Exceptions

- SQLException class - A SQL error occurred.

IsColumnGlobalAutoIncrement method

Checks whether the named column's default is set to global autoincrement.

Prototypes

```
' Visual Basic
Public Function IsColumnGlobalAutoIncrement( _
  ByVal name As String _
) As Boolean
```

```
// C#
public bool IsColumnGlobalAutoIncrement(
  string name
);
```

Parameters

- **name** name of the column.

Return value — true if the column is global autoincrementing, false if it is not global autoincrementing.

Exceptions
- SQLException class - A SQL error occurred.

See also
- "TableSchema class" on page 585
- "TableSchema members" on page 585
- "GetColumnPartitionSize method" on page 589
- "DatabaseID property" on page 348

IsColumnNewUUID method

Checks whether the named column's default is set to a new UUID (Guid).

Prototypes

```
' Visual Basic
Public Function IsColumnNewUUID( _
  ByVal name As String _
) As Boolean

// C#
public bool IsColumnNewUUID(
  string  name
);
```

Parameters
- **name** name of the column.

Return value — True if the column defaults to a new UUID, false if the column does not default to a new UUID.

Exceptions
- SQLException class - A SQL error occurred.

IsColumnNullable method

Checks whether the named column is nullable.

Prototypes

```
' Visual Basic
Public Function IsColumnNullable( _
  ByVal name As String _
) As Boolean

// C#
public bool IsColumnNullable(
  string  name
);
```

Parameters
- **name** name of the column.

Return value — true if the column is nullable, false if not nullable.

Exceptions
- SQLException class - A SQL error occurred.

IsInPublication method

Checks whether this table is contained in the named publication.

Prototypes

```
' Visual Basic
Public Function IsInPublication( _
  ByVal pubName As String _
) As Boolean

// C#
public bool IsInPublication(
  string  pubName
);
```

Parameters

- **pubName** name of the publication.

Return value

true if the table is in the publication, false if the table is not in the publication.

Exceptions

- SQLException class - A SQL error occurred.

Index

D

E

F

G

H

I

K

L

M

S

T

U

V