



**PUBLIC**

SQL Anywhere - UltraLite

Document Version: 17.01.0 – 2021-10-15

# UltraLite Administration

# Content

- 1 UltraLite - Database Management and Developer Guide . . . . . 7**
- 1.1 UltraLite Overview. . . . . 8
  - UltraLite Architecture. . . . . 9
  - UltraLite Synchronization Client Features. . . . . 10
  - UltraLite Supported Platforms. . . . . 11
  - UltraLite and SQL Anywhere Feature Comparisons. . . . . 11
  - UltraLite Database Limitations. . . . . 15
  - CustDB Sample Application Overview. . . . . 17
  - UltraLite Solution Considerations for Microsoft Windows Mobile. . . . . 20
- 1.2 UltraLite Security Considerations. . . . . 23
- 1.3 Configuring UltraLite Clients to Use Transport Layer Security. . . . . 25
- 1.4 UltraLite Database Creation Approaches. . . . . 26
  - Creating an UltraLite Database with the *Create Database Wizard*. . . . . 28
  - UltraLite Database Creation Using a Command Prompt. . . . . 29
  - UltraLite Database Creation Using a MobiLink Synchronization Model. . . . . 29
  - UltraLite Database Creation Through Central Administration of Remote Databases. . . . . 30
  - Creating an UltraLite Database from an XML File. . . . . 30
  - UltraLite Database Creation on a First Connection. . . . . 32
  - How to Access Creation Option Values. . . . . 32
  - UltraLite Character Sets. . . . . 32
  - Database Security. . . . . 35
- 1.5 Conversion from a SQL Anywhere Database to an UltraLite Database. . . . . 37
- 1.6 UltraLite Database Connections. . . . . 39
  - UltraLite Connection Strings and Parameters. . . . . 39
  - UltraLite Connection Parameters and the ULSQLCONNECT Environment Variable. . . . . 41
  - UltraLite File Path Formats in Connection Parameters. . . . . 42
- 1.7 UltraLite Database Tasks and Features. . . . . 43
  - Reading Database Properties. . . . . 43
  - Accessing Database Options. . . . . 45
  - UltraLite Event Notifications. . . . . 46
  - Isolation Levels. . . . . 48
  - Validating an UltraLite Database. . . . . 51
  - UltraLite Database Back up and Recovery. . . . . 52
- 1.8 UltraLite Database Schemas. . . . . 52
  - UltraLite Tables and Columns. . . . . 53
  - UltraLite Indexes. . . . . 62

	UltraLite Users. . . . .	66
1.9	UltraLite as a MobiLink Client. . . . .	72
	UltraLite Clients. . . . .	73
	Microsoft ActiveSync Synchronization Overview. . . . .	92
	UltraLite Synchronization Parameters. . . . .	93
	UltraLite Network Protocol Options. . . . .	122
1.10	UltraLite Deployment. . . . .	123
	UltraLite Application Build and Deployment Specifications. . . . .	124
	UltraLite Database Deployment Techniques. . . . .	130
	Deploying UltraLite Database Schema Upgrades. . . . .	131
	UltraLite Engine Startup. . . . .	133
	Registering Applications with the Microsoft ActiveSync Manager. . . . .	134
1.11	Tutorial: Building the UltraLite CustDB Sample Application. . . . .	135
	Lesson 1: Building and Running the CustDB Application. . . . .	136
	Lesson 2: Starting the MobiLink Server and Performing an Initial Synchronization. . . . .	137
	Lesson 3: Updating Data in the UltraLite Database. . . . .	138
	Lesson 4: Synchronizing the UltraLite Database with the Consolidated Database. . . . .	140
	Lesson 5: Browsing MobiLink Synchronization Scripts. . . . .	141
1.12	UltraLite Database Reference. . . . .	143
	UltraLite Options. . . . .	143
	UltraLite Connection Parameters. . . . .	181
	UltraLite Database Properties. . . . .	203
	UltraLite Database Options. . . . .	206
	UltraLite Utilities. . . . .	212
	UltraLite System Tables. . . . .	248
1.13	UltraLite SQL reference. . . . .	254
	UltraLite SQL Language Elements. . . . .	255
	SQL Data Types. . . . .	288
	Spatial Data Types. . . . .	320
	User-defined Data Types and Their Equivalents. . . . .	322
	SQL Functions. . . . .	323
	UltraLite SQL Statements. . . . .	516
1.14	UltraLite Performance Tips. . . . .	569
	Cache Size Adjustment for an UltraLite Database. . . . .	569
	Query Performance Tips. . . . .	570
	Insert and Update Performance Tips. . . . .	582
	UltraLite Benchmark Tips. . . . .	587
1.15	UltraLite Troubleshooting. . . . .	592
	Unable to Start the UltraLite Engine. . . . .	593
	Unable to Connect to Databases After Upgrade. . . . .	593
	UltraLite Database Corruption. . . . .	594

	Database Size Not Stabilizing. . . . .	595
	Importing ASCII Data into a New UltraLite Database. . . . .	596
	Utilities Still Running as the Previous Version. . . . .	597
	Result Set Changes Unpredictably. . . . .	598
	UltraLite Engine Client Fails with Error -764. . . . .	598
<b>2</b>	<b>UltraLite.NET Application Development. . . . .</b>	<b>600</b>
2.1	UltraLite .NET System Requirements and Supported Platforms. . . . .	601
2.2	SQL Anywhere Tools in Microsoft Visual Studio. . . . .	602
2.3	Connection Setup for an UltraLite Database. . . . .	602
	Connecting to an UltraLite Database Using UltraLite.NET. . . . .	603
2.4	Data Creation and Modification in UltraLite.NET Using SQL Statements. . . . .	604
	Data Modification in UltraLite.NET Using INSERT, UPDATE, and DELETE. . . . .	605
	Retrieving Data in UltraLite.NET Using SELECT. . . . .	608
	Result Set Schema Description. . . . .	609
	SQL Result Set Navigation in UltraLite.NET. . . . .	609
2.5	Data creation and modification in UltraLite.NET using the UTable Class. . . . .	610
	Row Navigation in UltraLite.NET. . . . .	611
	UltraLite Modes. . . . .	612
	Row Insertion in UltraLite.NET. . . . .	612
	Row Updates. . . . .	613
	Row Searches. . . . .	614
	Row Retrieval. . . . .	616
	Row Deletions in UltraLite.NET. . . . .	617
2.6	Transaction Management in UltraLite.NET. . . . .	618
2.7	Schema Information in UltraLite.NET. . . . .	618
2.8	Error Handling in UltraLite.NET. . . . .	619
2.9	MobiLink Data Synchronization in UltraLite.NET. . . . .	620
	Synchronization Initiation in a C# Application. . . . .	620
	Microsoft ActiveSync Synchronization Setup in UltraLite.NET. . . . .	621
2.10	How to Deploy UltraLite.NET Applications. . . . .	621
	Deploying an UltraLite.NET Application for Microsoft Windows Mobile. . . . .	622
	Deploying an UltraLite.NET Application for Windows Mobile (UltraLite Engine). . . . .	623
2.11	Tutorial: Building a Microsoft Windows Mobile Application Using UltraLite.NET. . . . .	625
	Lesson 1: Creating a Microsoft Visual Studio Project. . . . .	626
	Lesson 2: Creating an UltraLite Database. . . . .	629
	Lesson 3: Adding Database Connection Controls to the Application. . . . .	631
	Lesson 4: Inserting, Updating, and Deleting Data. . . . .	633
	Lesson 5: Building and Deploying the Application. . . . .	637
	Code Listing for C# Tutorial. . . . .	639
	Code Listing for Microsoft Visual Basic Tutorial. . . . .	641

<b>3</b>	<b>UltraLite - C++ Programming.</b>	<b>643</b>
3.1	System Requirements and Supported Platforms.	643
3.2	UltraLite Application Development Using C++.	644
	UltraLite C++ Application Development.	644
	UltraLite C++ Application Development Using Embedded SQL.	676
	UltraLite Application Development for Microsoft Windows Mobile.	708
3.3	Tutorial: Building a Windows Application using the C++ API.	717
	Lesson 1: Creating and Connecting to a Database.	718
	Lesson 2: Inserting Data into the Database.	721
	Lesson 3: Selecting and Listing Rows from the Table.	723
	Lesson 4: Adding Synchronization to Your Application.	725
	Reviewing the Code Listing for the Tutorial.	726
3.4	API Reference.	728
	UltraLite C++ Common API Reference.	729
	UltraLite C++ API Reference.	731
	UltraLite Embedded SQL API Reference.	731
<b>4</b>	<b>UltraLite - Java Programming.</b>	<b>734</b>
4.1	System Requirements and Supported Platforms.	734
4.2	UltraLiteJ Application Development.	734
	Quick Start Guide to UltraLiteJ Application Development.	736
	Android Setup Considerations.	736
	UltraLite Database Creation and Connection Approaches.	737
	Quick Start Guide to Schema Operations and Data Management.	739
	Schema Information in UltraLiteJ.	748
	Error Handling in UltraLiteJ.	749
	MobiLink Data Synchronization Using UltraLiteJ.	750
	Deploying an UltraLiteJ application for Android.	751
	Code Examples.	753
4.3	Tutorial: Building an Android Application.	753
	Lesson 1: Setting up a New Android Project.	755
	Lesson 2: Starting the MobiLink Server.	757
	Lesson 3: Running Your Android Application.	758
	Lesson 4: Testing Your Android Application and Synchronizing.	759
	Lesson 5: Cleaning up.	761
4.4	UltraLiteJ API Reference.	761
<b>5</b>	<b>UltraLite - UWP Programming.</b>	<b>763</b>
5.1	System Requirements and Supported Platforms.	763
5.2	UltraLite for UWP Application Development.	764
	Quick Start Guide to UltraLite for UWP Application Development.	765
	UltraLite for UWP Setup Considerations.	765

	Quick Start Guide to Schema Operations and Data Management. . . . .	766
	Deploying an UltraLite Application for Windows Phone or Windows Store Apps. . . . .	769
5.3	Tutorial: Building a Windows Phone Application. . . . .	770
	Lesson 1: Setting up a New Windows Phone Application. . . . .	771
	Lesson 2: Starting the MobiLink Server. . . . .	772
	Lesson 3: Running Your Windows Phone Application and Synchronizing. . . . .	773
	Lesson 4: Cleaning up. . . . .	774
5.4	UltraLite for UWP API Reference. . . . .	775

# 1 UltraLite - Database Management and Developer Guide

This book describes the UltraLite database systems for small devices.

## In this section:

### [UltraLite Overview \[page 8\]](#)

UltraLite is a compact relational database management system with many of the same features as SQL Anywhere. It can be used to create mobile databases for small-footprint devices such as smartphones, handheld computers, and tablet PCs.

### [UltraLite Security Considerations \[page 23\]](#)

Because databases may contain proprietary, confidential, or private information, ensuring that the database, the data, and communications over networks are designed for security is very important.

### [Configuring UltraLite Clients to Use Transport Layer Security \[page 25\]](#)

UltraLite clients can be configured to use transport layer security over a TCP/IP or HTTPS protocol.

### [UltraLite Database Creation Approaches \[page 26\]](#)

There are three common types of database creation methods:

### [Conversion from a SQL Anywhere Database to an UltraLite Database \[page 37\]](#)

Create an UltraLite database from a SQL Anywhere reference database by running the ulinit utility with the -a option. The new database is created with the same settings as those in the reference database where possible.

### [UltraLite Database Connections \[page 39\]](#)

Applications that use a database must establish a connection to that database before transactions can occur. By connecting to an UltraLite database, you form a channel through which all activity from the application takes place.

### [UltraLite Database Tasks and Features \[page 43\]](#)

There are many tasks you perform and features you can use to manage UltraLite databases.

### [UltraLite Database Schemas \[page 52\]](#)

The logical framework of the database is known as a **schema**.

### [UltraLite as a MobiLink Client \[page 72\]](#)

You can configure UltraLite to act as a MobiLink client.

### [UltraLite Deployment \[page 123\]](#)

In the majority of cases, development occurs on a Windows desktop or macOS with the final release target for UltraLite being the mobile device.

### [Tutorial: Building the UltraLite CustDB Sample Application \[page 135\]](#)

In this tutorial you learn how to run the MobiLink server to carry out data synchronization between the consolidated database and the UltraLite remote, use SQL Central to browse the data in the UltraLite remote, and manage UltraLite databases with UltraLite utilities.

### [UltraLite Database Reference \[page 143\]](#)

UltraLite provides many tools and features to help you run, manage, and configure UltraLite databases.

[UltraLite SQL reference \[page 254\]](#)

UltraLite supports many SQL language features and elements.

[UltraLite Performance Tips \[page 569\]](#)

Several topics are provided to help improve performance of UltraLite databases.

[UltraLite Troubleshooting \[page 592\]](#)

Several topics are provided to help you troubleshoot problems with your UltraLite database.

## 1.1 UltraLite Overview

UltraLite is a compact relational database management system with many of the same features as SQL Anywhere. It can be used to create mobile databases for small-footprint devices such as smartphones, handheld computers, and tablet PCs.

UltraLite includes a built-in synchronization client that tracks changes made in UltraLite databases, and exchanges updates with a MobiLink server over a network. As a MobiLink client, UltraLite ensures that mobile applications can stay synchronized with a central database and with other UltraLite databases.

In UltraLite, the database management systems that are typically found in a database server are implemented as an in-process runtime library. The runtime library and the application are part of the same process.

### In this section:

[UltraLite Architecture \[page 9\]](#)

UltraLite supports several mobile platforms and consists of API development, database management, and database layers.

[UltraLite Synchronization Client Features \[page 10\]](#)

UltraLite includes a built-in bi-directional synchronization client that causes all data in an UltraLite database to be synchronized by default.

[UltraLite Supported Platforms \[page 11\]](#)

UltraLite supports various mobile platforms. Third-party software is required for UltraLite database development.

[UltraLite and SQL Anywhere Feature Comparisons \[page 11\]](#)

The availability and functionality of features can differ between UltraLite and SQL Anywhere.

[UltraLite Database Limitations \[page 15\]](#)

There are several hard limits that apply to UltraLite databases.

[CustDB Sample Application Overview \[page 17\]](#)

The CustDB sample is a multi-tiered database management solution that implements MobiLink synchronization with a SQL Anywhere consolidated database.

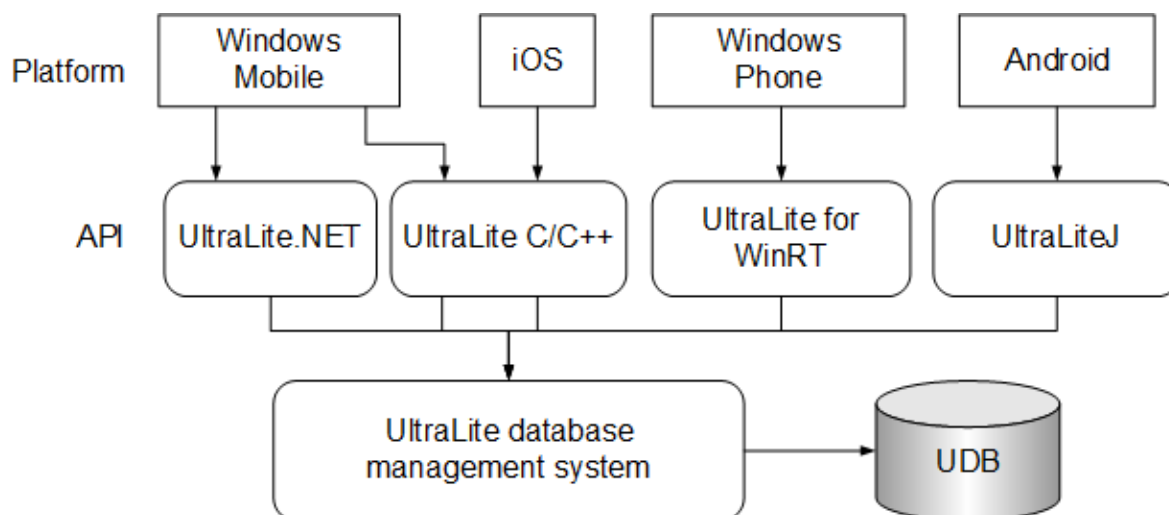
[UltraLite Solution Considerations for Microsoft Windows Mobile \[page 20\]](#)

There are several UltraLite design options that are available for Microsoft Windows mobile development.



## 1.1.1 UltraLite Architecture

UltraLite supports several mobile platforms and consists of API development, database management, and database layers.



### Mobile platform support

Your target mobile platform determines which UltraLite API is available for application development.

### API development layer

Refer to the diagram above to determine which API to use for your target mobile platform.

### Database management layer and synchronization client

Use the UltraLite APIs to interface with the UltraLite database management system. This system allows you to create and connect to an UltraLite database.

A comprehensive set of administration tools is provided to help you maintain your UltraLite project. You can run these tools as either command line utilities or wizards in the UltraLite plug-in for SQL Central.

### Database layer

This layer is the local data repository stored as a file. UltraLite databases are stored as UDB files. UDB files are portable across all mobile platforms. UltraLite databases don't contain information about the distribution of data within the database. UltraLite keeps track of its transactions internally, not in a separate log file. The UltraLite temporary file is stored in the same directory as the database file.

## Related Information

[UltraLite C++ Application Development \[page 644\]](#)

[UltraLite.NET Application Development \[page 600\]](#)

[UltraLite for UWP Application Development \[page 764\]](#)

[UltraLite Utilities \[page 212\]](#)

[UltraLite Database Limitations \[page 15\]](#)

## 1.1.2 UltraLite Synchronization Client Features

UltraLite includes a built-in bi-directional synchronization client that causes all data in an UltraLite database to be synchronized by default.

Users new to MobiLink synchronization may use this default behavior until business requirements necessitate a custom synchronization design to alter what UltraLite data gets synchronized to the consolidated database. Unlike SQL Anywhere remote databases, you do not need to increase the size of the UltraLite footprint to include synchronization functionality.

Important synchronization features built into the UltraLite runtime include a row state tracking mechanism and a synchronization state tracking mechanism.

### The Row State Tracking Mechanism

Tracking the state of tables and rows is particularly important for data synchronization. Each row in an UltraLite database has an associated row state structure. In addition to synchronization, UltraLite also uses the row states to control transaction processing and data recovery.

### Synchronization State Tracking

UltraLite uses a progress counter to ensure robust synchronization. Each upload is given a unique number to identify it. This allows UltraLite to determine whether an upload was successful when a communication error occurs.

When you first create a new database, UltraLite always sets the synchronization progress counter to zero. A progress counter value of zero identifies the database as a new UltraLite database, which tells the MobiLink server to reset its state information for this client.

#### Caution

Because UltraLite increments the progress counter each time a synchronization occurs, you cannot synchronize an UltraLite database to different consolidated databases. If the progress counter value is not zero and does not match that sequence number stored in the consolidated database, MobiLink synchronization reports an offset mismatch and synchronization fails. You cannot replace an UltraLite database with a backup copy if the progress counter is older than the current value.

## Related Information

[UltraLite Database Row State Management \[page 584\]](#)

[MobiLink Synchronization](#)

[Tutorial: Using MobiLink with a SQL Anywhere Consolidated Database](#)

[UltraLite as a MobiLink Client \[page 72\]](#)

## 1.1.3 UltraLite Supported Platforms

UltraLite supports various mobile platforms. Third-party software is required for UltraLite database development.

## Related Information

[UltraLite Network Protocol Options \[page 122\]](#)

[Supported Platforms](#)

[UltraLite as a MobiLink Client \[page 72\]](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

## 1.1.4 UltraLite and SQL Anywhere Feature Comparisons

The availability and functionality of features can differ between UltraLite and SQL Anywhere.

### i Note

The UltraLite database management system adds 750-1500 KB to the size of your application. The SQL Anywhere database, database server, and synchronization client add approximately 6 MB.

Feature	SQL Anywhere	UltraLite	Considerations
Transaction processing, and multi-table joins	X	X	
Triggers, stored procedures, and views	X		
External stored procedures (callable external DLLs)	X		
Built-in referential and entity integrity	X	X	

Feature	SQL Anywhere	UltraLite	Considerations
Cascading updates and deletes	X	Limited	Declarative referential integrity, where deletes and updates are cascaded, is a feature that is not supported in UltraLite databases, except during synchronization when deletes are cascaded for this purpose.
Dynamic, multiple database support	X	X	
Multithreaded application support	X	X	
Row-level locking	X	X	
XML unload and load utilities		X	UltraLite uses ulload, ulunload, uljload, and uljunload administration tools to complete XML load and unloads.
SQLX functionality	X		
SQL functions	X	X	Not all SQL functions are available for use in UltraLite applications. If you use an unsupported function, you trigger an error.
SQL statements	X	X	The scope of SQL statements is different compared to SQL Anywhere.
Integrated HTTP server	X		
Strong encryption for database files and network communications	X	X	
Event scheduling and handling	X	X	An UltraLite event model differs from SQL Anywhere.
High-performance, self-tuning, cost-based query optimizer	X		UltraLite has a query optimizer that is not as extensive as that of SQL Anywhere.
Choice of several thread-safe APIs	X	X	UltraLite gives application developers a uniquely flexible architecture that allows for the creation of applications for changing and/or varied deployment environments.
Cursor support	X	X	

Feature	SQL Anywhere	UltraLite	Considerations
Dynamic cache sizing	X	X	UltraLite allows you to set an initial, minimum, and maximum cache size for a database using the <code>CACHE_SIZE</code> , <code>CACHE_MIN_SIZE</code> , and <code>CACHE_MAX_SIZE</code> connection parameters, respectively. The size of the cache is optimized by UltraLite on an ongoing basis, up to the maximum size (if specified).
Database recovery after system or application failure	X	X	
Binary Large Object (BLOB) support	X	X	UltraLite cannot index or compare BLOBs.
Microsoft Microsoft Windows Performance Monitor integration	X		
Online table and index defragmentation	X		
Online backup	X		
Direct device connections to a Microsoft Windows Mobile device from the desktop.		X	SQL Anywhere databases need a database server before allowing desktop connections to the database that you deploy on a Microsoft Windows Mobile device. On UltraLite, you prefix the connection string with <code>WCE:\</code> .
High-performance updates and retrievals through the use of indexes	X	X	UltraLite uses a mechanism to determine whether each table is searched using an index or by scanning the rows directly.  Additionally, you can hash indexes to speed up data retrieval.  You can use the <code>max_hash_size</code> creation parameter to set the maximum hash size.

Feature	SQL Anywhere	UltraLite	Considerations
Synchronizing to SAP HANA, Oracle, DB2, SAP Adaptive Server Enterprise, Microsoft SQL Server, MySQL, or SQL Anywhere	X	X	
Built-in synchronization		X	Unlike SQL Anywhere deployments, UltraLite does not require a client agent for synchronization. Synchronization is built into the UltraLite runtime to minimize the components you must deploy.
In-process execution		X	
Computed columns	X		
Declared temporary tables/global temporary tables	X		
System functions	X		
Timestamp columns	X	X	SQL Anywhere supports the DEFAULT TIMESTAMP default.  UltraLite only supports the DEFAULT CURRENT TIME- STAMP default. Therefore, UltraLite can not automatically update the timestamp when the row is updated.
User-based permission scheme to determine object-based ownership and access	X		UltraLite is primarily designed for single user databases in which an authorization system is not needed. However, you can include up to four user IDs and passwords, which are used for authentication purposes only. These users have access to all database objects.
Spatial data	X	Limited	UltraLite supports point data only.
Full text data	X		

## Related Information

[Avoiding Synchronization Issues with Foreign Key Cycles \[page 85\]](#)

- [UltraLite SQL Statements \[page 516\]](#)
- [Benefits of UltraLite APIs for Microsoft Windows Mobile \[page 21\]](#)
- [UltraLite Database Limitations \[page 15\]](#)
- [UltraLite File Path Formats in Connection Parameters \[page 42\]](#)
- [UltraLite Users \[page 66\]](#)
- [UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)
- [UltraLite Database Unload Utility \(ulunload\) \[page 243\]](#)
- [UltraLite max\\_hash\\_size Creation Option \[page 161\]](#)

## 1.1.5 UltraLite Database Limitations

There are several hard limits that apply to UltraLite databases.

In some cases, the limits are beyond the maximum capabilities of mobile devices. Performance considerations and device capabilities impose other limitations.

Item	UltraLite database limitations
Database and file size	16 GB for 4, 8, or 16 KB page size. 8 GB for 2 KB page size. 1 GB for 1 KB page size. Less if there is an operating system limit on file size.
Temporary file size	16 GB for 4, 8, or 16 KB page size. 8 GB for 2 KB page size. 1 GB for 1 KB page size. Less if there is an operating system limit on file size.
Cache size	Limited by the available memory on the device, up to 64 K pages.
Dynamic cache sizing	<p>UltraLite allows you to set an initial, minimum, and/or maximum cache size for a database. The size of the cache is optimized by UltraLite on an ongoing basis, up to the maximum size (if specified).</p> <p>The CACHE_SIZE, CACHE_MIN_SIZE, CACHE_MAX_SIZE connection parameters can be used to set and adjust the cache size.</p>
Maximum number of concurrent open connections supported by a database	Up to 14.
Maximum number of concurrent open connections to all databases	Limited only by memory.
Maximum number of databases that can run concurrently	Limited only by memory.
Maximum number of applications that can connect to a database concurrently	Use the UltraLite engine to handle multiple concurrent applications connecting to the database. Otherwise, only one application can connect to a database at one time.
Returned SQL function values	In some cases, UltraLite limits expression results to 2000 bytes.

Item	UltraLite database limitations
Rows per table	<p>Up to 16 million.</p> <p>Sometimes changes to the row (deletes and updates) and other state information are maintained with the row data. This information allows those changes to be synchronized. So, the actual row limit can be smaller than 16 million, depending on the number of transactions on a table between synchronization.</p>
Row size	<p>The length of each packed row must not exceed the page size.</p> <p>Character strings are stored without padding when they are shorter than the column size. This restriction excludes columns declared as LONG BINARY and LONG VARCHAR as these strings are stored separately.</p>
Rows per database	Limited by database size.
Table size	Limited by the database size.
Tables per database	Limited by the database size.
Columns per table	Row size is limited by page size, so the practical limit on the number of columns per table is derived from this size. Typically, this practical limit is much less than 4000.
Indexes per table	Limited by the database size.
Number of publications	Up to 63.
Database page size	Minimum 1 KB; up to 16 KB.
Cursors per connection	The maximum number of allowable cursors on a given connection to an UltraLite database is 64 (all platforms).
Strings	The row must fit on a page.
Binary data types	The row must fit on a page.
Long binary/long varchar size	Limited only by database size.
Blob size	Limited by file size.
Available Isolation levels	0 (read uncommitted) or 1 (read committed).
Cascading updates and deletes	Declarative referential integrity, where deletes and updates are cascaded, is a feature that is not supported in UltraLite databases, except during synchronization when deletes are cascaded for this purpose.
Event scheduling and handling	An UltraLite event model differs from SQL Anywhere.

## Related Information

[UltraLite Transaction Processing \[page 585\]](#)

[Row Packing and Table Definitions \[page 54\]](#)



[Physical Limitations on Size and Number of Databases](#)  
[UltraLite CACHE\\_SIZE Connection Parameter \[page 187\]](#)  
[UltraLite CACHE\\_MIN\\_SIZE Connection Parameter \[page 186\]](#)  
[UltraLite CACHE\\_MAX\\_SIZE Connection Parameter \[page 185\]](#)  
[UltraLite Database Unload Utility \(ulunload\) \[page 243\]](#)  
[UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

## 1.1.6 CustDB Sample Application Overview

The CustDB sample is a multi-tiered database management solution that implements MobiLink synchronization with a SQL Anywhere consolidated database.

CustDB is installed with SQL Anywhere and consists of the following:

- A **consolidated** SQL Anywhere database. The database is pre-populated with sales status data.
- A **remote** UltraLite database. This database is initially empty.
- An UltraLite client application.
- MobiLink server synchronization scripts.

### i Note

You can only run one instance of CustDB at a time. Trying to run more than one instance brings the first instance to the foreground.

CustDB allows sales personnel to track and monitor transactions and then pool information from two types of users:

- Sales personnel that authenticate with user IDs 51, 52, and 53.
- Mobile managers that authenticate with user ID 50.

Information gathered by these different users can be synchronized with the consolidated database.

Both the consolidated and remote databases contain a table named ULOrder. While the consolidated database holds all orders (approved and those pending approval), the UltraLite remote database only displays a subset of rows according to the user that has authenticated.

Columns in the table appear as fields in the client application. When you add an order, you must populate the Customer, Product, Quantity, Price, and Discount fields. You can also append other details such as Status or Notes. The timestamp column identifies whether the row needs to be synchronized.

The synchronization logic for CustDB is held in the consolidated database as MobiLink synchronization scripts. Synchronization logic allows you to determine how much of the consolidated database you need to download and/or upload. You can download complete tables or partial tables (with either row or column subsets) using such techniques as timestamp-based synchronization or snapshot synchronization.

You can use SQL Central to browse the synchronization scripts that are stored in the consolidated database. SQL Central is the primary tool for adding scripts to the database.

The `custdb.sql` file adds each synchronization script to the consolidated database by calling `ml_add_connection_script` or `ml_add_table_script`. Connection scripts control high-level events that are not associated with a particular table. Use these events to perform global tasks that are required during every

synchronization. Table scripts allow actions at specific events relating to the synchronization of a specific table, such as the start or end of uploading rows, resolving conflicts, or selecting rows to download.

## SQL Anywhere CustDB database

This is the consolidated database. During installation, an ODBC data source called SQL Anywhere 17 CustDB is created for this database. The database file is located at `%SQLANYSAMPI7%\UltraLite\CustDB\`.

You can erase changes that were synchronized into the consolidated `CustDB.db` file, so you have a clean version to work with using this script: `%SQLANYSAMPI7%\UltraLite\CustDB\makedbs.cmd`.

## The UltraLite CustDB Database

This is the remote version of the consolidated database that contains only a subset of the information, depending on which user synchronizes the database.

The file name and location can vary depending on the platform, programming language, or even device.

- For UltraLite.NET: `%SQLANYSAMPI7%\UltraLite.NET\CustDB\Common\custdb.udb`
- For all other platforms and APIs: `%SQLANYSAMPI7%\UltraLite\CustDB\custdb.udb`

## RDBMS-Specific Build Scripts

The SQL scripts rebuild a CustDB consolidated database for any one of the supported RDBMSs.

In the `%SQLANYSAMPI7%\MobiLink\CustDB` directory, you can find the following files:

- For SQL Anywhere: `custdb.sql`
- For Adaptive Server Enterprise: `custase.sql`
- For Microsoft Azure: `custmss.sql`
- For Microsoft SQL Server: `custmss.sql`
- For Oracle: `custora.sql`
- For IBM DB2: `custdb2.sql`

### **i** Note

Support for IBM DB2 consolidated databases is deprecated.

- For MySQL: `custmys.sql`

## UltraLite CustDB Client Applications and ReadMe Files

These are the end-user applications that provide a user-friendly interface to the UltraLite remote database. There is a sample client installed for each supported platform.

Each client application also contains a `ReadMe.html` or `ReadMe.txt` file. Each file includes an outline of the steps that are required to build and run the sample.

The location of the application and its ReadMe depends on your development environment.

## Synchronization Logic

The UltraLite database SQL statements and synchronization calls are located in `custdbcpp.cpp` for the C++ API.

### In this section:

[CustDB File Locations for UltraLite \[page 19\]](#)

The CustDB application is built for many development environments.

## Related Information

[CustDB Sample for MobiLink](#)

[CustDB Consolidated Database Setup](#)

[MobiLink Consolidated Databases](#)

[Lesson 1: Building and Running the CustDB Application \[page 136\]](#)

### 1.1.6.1 CustDB File Locations for UltraLite

The CustDB application is built for many development environments.

## UltraLite for Microsoft Windows 32-Bit Desktop

You do not need to build the CustDB application before running it.

You can find the CustDB executable file in the `%SQLANY17%\UltraLite\Windows\x86` directory.

## UltraLite for C++

### All versions of C++

You can find multiple versions of the C++ CustDB project file because of the many C++ development environments. Most versions use the generic files. These files are located in the [C:\Program Files\SQL Anywhere 17\UltraLite\Custdb](#) directory.

For information about all versions of C++ CustDB applications, see [C:\Program Files\SQL Anywhere 17\UltraLite\Custdb\readme.txt](#).

### Microsoft Visual Studio

You can find project files in the [%SQLANYSAMP17%\UltraLite\CustDB](#) directory.

## UltraLite.NET

You can find project files specific to Microsoft Visual Studio in the [C:\Users\Public\Documents\SQL Anywhere 17\Samples\UltraLite.NET\CustDB](#) directory.

## 1.1.7 UltraLite Solution Considerations for Microsoft Windows Mobile

There are several UltraLite design options that are available for Microsoft Windows mobile development.

### UltraLite API Selection

The benefits of using each of the following APIs for Microsoft Windows Mobile development are described:

- UltraLite C++
- UltraLite Embedded SQL
- UltraLite.NET

### Data Management Component Selection

The benefits of using each of the following data management components for Microsoft Windows Mobile development are described:

- UltraLite in-process runtime environment
- UltraLite database engine

### In this section:

#### [Benefits of UltraLite APIs for Microsoft Windows Mobile \[page 21\]](#)

The UltraLite C++, Embedded SQL, and .NET APIs offer several data access models, including a simple table-based data access interface and dynamic SQL for more complex queries.

#### [UltraLite Data Management Components for Microsoft Windows Mobile \[page 21\]](#)

UltraLite allows you to build a small-footprint relational database solution without requiring the additional overhead of setting up a separate database server.

## 1.1.7.1 Benefits of UltraLite APIs for Microsoft Windows Mobile

The UltraLite C++, Embedded SQL, and .NET APIs offer several data access models, including a simple table-based data access interface and dynamic SQL for more complex queries.

By combining these benefits, UltraLite gives application developers a flexible architecture for creating applications for their varied deployment environments.

### UltraLite.NET API Benefits

The UltraLite.NET API is usually recommended for Microsoft Windows Mobile development because the SQL Anywhere .NET API provides common programming models that are shared between UltraLite components and SQL Anywhere, and because of the .NET programming compared C++.

### UltraLite C++ and Embedded SQL API Benefits

While UltraLite provides high performance in a variety of environments and use cases, Embedded SQL and the UltraLite C++ API are the lowest level APIs and generally deliver the highest performance.

Use the UltraLite C++ API when you are trying to create the smallest application footprint. These applications typically yield the best performance and still maintain a small application file size.

### Related Information

[Appendix - .NET Framework](#)

[UltraLite C++ Application Development \[page 644\]](#)

[UltraLite C++ Application Development Using Embedded SQL \[page 676\]](#)

[UltraLite.NET Application Development \[page 600\]](#)

## 1.1.7.2 UltraLite Data Management Components for Microsoft Windows Mobile

UltraLite allows you to build a small-footprint relational database solution without requiring the additional overhead of setting up a separate database server.

UltraLite programming interfaces use one of two approaches: the **UltraLite in-process runtime library** and the **UltraLite database engine**. Both approaches control connection and data access requests.

Both components include a built-in bi-directional synchronization client that links UltraLite databases with the MobiLink synchronization server.

## UltraLite In-Process Runtime Library

The UltraLite in-process runtime is recommended when only one application needs to access a database at a time.

The runtime and the application are part of the same process, which makes the database specific to the application. The runtime library manages UltraLite databases and built-in synchronization operations.

Linking to the runtime requires a different import library/DLL pair from that of the engine.

UltraLite supports both static and dynamic linkage.

### Static linking

Static linking requires less device memory and is more effective when only a single UltraLite application is used on the device.

### Dynamic linking

Dynamic linking may be more economical with device memory when multiple UltraLite applications are used on the device.

## UltraLite Database Engine (The `uleng17.exe` Utility)

The UltraLite engine is only available for Microsoft Windows desktop and Microsoft Windows Mobile platforms. The engine is a separate executable that uses the UltraLite runtime library and supports concurrent access from multiple applications. Each application must use a client library to communicate with the UltraLite engine.

The UltraLite engine requires more system resources than the UltraLite runtime and may yield lower performance when large amounts of data are moved between the client and database.

Connecting to the engine requires that you specify a different import library/DLL pair than that of the runtime.

The UltraLite engine is required under the following conditions:

- Multiple processes access the same database file at potentially the same time (same time means multiple processes have connections open to the same database at the same time).
- Central Administration is used to manage the UltraLite application database.

## Related Information

[UltraLite Synchronization Client Features \[page 10\]](#)

[How to Build and Deploy UltraLite C++ Applications \[page 667\]](#)

[UltraLite Engine Startup \[page 133\]](#)

[UltraLite Concurrency \[page 583\]](#)

[UltraLite and SQL Anywhere Feature Comparisons \[page 11\]](#)

[UltraLite Database Limitations \[page 15\]](#)

[UltraLite Application Build and Deployment Specifications \[page 124\]](#)

[UltraLite Engine Utility \(uleng17\) \[page 223\]](#)

## 1.2 UltraLite Security Considerations

Because databases may contain proprietary, confidential, or private information, ensuring that the database, the data, and communications over networks are designed for security is very important.

Several features are included to assist in building a secure environment for your data.

### Database Encryption

By default, databases are not encrypted or obfuscated when they are created. Text and binary columns can be read when using a viewing tool such as a hex editor. UltraLite provides the following database creation methods:

#### AES 256-bit encryption

This option encrypts databases with an AES 256-bit algorithm. Strong encryption provides security against skilled and determined attempts to gain access to the data. You do not need any special configuration to use AES encryption on your device.

#### FIPS 140-2 certified AES 256-bit encryption

Encryption libraries certified to comply to the FIPS 140-2 computer security standard *Security Requirements for Cryptographic Modules* are provided under a separate license. FIPS-certified AES encryption requires that you configure your device appropriately.

Encryption keys should contain a combination of characters, numbers, and special symbols to be effective. Long encryption keys reduce the chances of someone guessing the key.

#### **i** Note

After the database is encrypted, the encryption key cannot be recovered.

Using SQL Central wizards, you can specify UltraLite database encryption options during creation by clicking the *Encrypt the database* option and then clicking *Use strong encryption*. Select one of the AES algorithms and then enter an encryption key.

Using the ulinit utility, you can specify encryption using the -e option. Use the --fips option to specify whether to use FIPS-certified encryption. Specify the encryption key with the -k (--key) option.

UltraLite API encryption options are available when creating a database.

#### **⚠** Caution

You can change the encryption key after the database has been created but only under extreme caution.

This operation is costly and is non-recoverable. You can lose your database entirely if your operation terminates mid-course.

For strongly encrypted databases, store a copy of the key in a safe location. If you lose the encryption key, there is no way to access the data, even with the assistance of Technical Support. The database must be discarded and you must create a new database.

The DBKEY parameter must be supplied when connecting to the database; otherwise, the connections fail. Encryption keys should be treated as sensitive information.

## Database Obfuscation

This option provides protection against casual attempts to access data in the database but does not provide as much security as strong encryption. Obfuscation has a minimal performance impact. You do not need any special configuration to use simple obfuscation on your device.

### i Note

Consider the effects of database cache size when choosing to encrypt or obfuscate databases. There is an overhead increase between 5-10%, which can result in decreased performance. The precise effect on performance depends on the size of your cache. If your cache is too small, encryption can add significant overhead. However, if your cache is sufficiently large, you may not experience any difference at all.

To obfuscate data, specify `obfuscate=1` as a database creation parameter when you create your database. End users do not need to supply a corresponding connection parameter.

To obfuscate data with the UltraLiteJ API, use the `ConfigPersistent.enableObfuscation` method during database creation.

## Transport Layer Security (TLS)

MobiLink transport layer security is an inherent feature of the MobiLink HTTPS protocol. When using HTTPS and UltraLite clients, you can specify trusted certificates and certificate fields using network protocol options. There are two ways to specify trusted root certificates: using the UltraLite Initialize Database utility (`ulinit`) or the `trusted_certificates` protocol option. You can also specify client-side certificate information when using the UltraLite Load XML to Database utility (`ulload`).

## End-to-End Encryption for macOS and iOS

To use end-to-end encryption when synchronizing macOS and iOS UltraLite clients with a MobiLink server, you must encapsulate your public keys in a PEM encoded X509 certificate (as opposed to a PEM public key file) and supply an E2EE private key. To create a PEM encoded X509 certificate with an E2EE private key, use the certificate creation utility, `createcert`.

## Related Information

[UltraLite Database Creation Approaches \[page 26\]](#)

[Cache Size Adjustment for an UltraLite Database \[page 569\]](#)



[UltraLite Network Protocol Options \[page 122\]](#)  
[Apple iOS and macOS Considerations \[page 646\]](#)  
[Separately Licensed Components](#)  
[Configuring UltraLite Clients to Use Transport Layer Security \[page 25\]](#)  
[UltraLite obfuscate Creation Option \[page 165\]](#)  
[UltraLite fips Creation Option \[page 159\]](#)  
[UltraLite Database Properties \[page 203\]](#)  
[UltraLite DBKEY Connection Parameter \[page 192\]](#)  
[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)  
[Certificate Creation Utility \(createcert\)](#)  
[UltraLite Load XML to Database Utility \(uload\) \[page 234\]](#)

## 1.3 Configuring UltraLite Clients to Use Transport Layer Security

UltraLite clients can be configured to use transport layer security over a TCP/IP or HTTPS protocol.

### Prerequisites

You must be using a TCP/IP or HTTPS protocol.

### Context

MobiLink transport layer security is an inherent feature of the MobiLink HTTPS protocol. If you use HTTPS and UltraLite clients, you can specify trusted certificates and certificate fields directly as network protocol options. There are two ways to specify trusted root certificates: using the UltraLite Initialize Database utility or the `trusted_certificates` protocol option.

### Procedure

1. Specify the TCP/IP or HTTPS protocol for synchronization. The keyword for secure TCP/IP is `tls`.
2. Specify TCP/IP or HTTPS protocol options.

The `certificate_company`, `certificate_unit`, and `certificate_name` protocol options are used to verify certificate fields.

You can also specify the `trusted_certificates` HTTPS protocol option, which overrides any trusted certificate information embedded in the UltraLite database.

## Results

The UltraLite client is configured to use transport layer security over either an HTTPS or TCP/IP protocol.

## Example

The following example is in C/C++ UltraLite. To specify tls, change HTTPS to tls.

```
auto ul_sync_info synch_info;
conn.InitSynchInfo( &synch_info );
synch_info.user_name = UL_TEXT( "50" );
synch_info.version = UL_TEXT( "ul_default" );
...
synch_info.stream = "https";
...
```

```
auto ul_sync_info synch_info;
...
synch_info.stream = "https";
synch_info.stream_parms = TEXT(
    "port=9999;
    certificate_company=SAP;
    certificate_unit=IAS;
    certificate_name=MobiLink");
```

```
auto ul_sync_info synch_info;
...
synch_info.stream = "https";
synch_info.stream_parms = TEXT(
    "port=9999;
    trusted_certificates=\rsaroot.crt;
    certificate_company=SAP;
    certificate_unit=IAS;
    certificate_name=MobiLink");
```

## Related Information

[MobiLink Client/Server Communications Encryption](#)  
[MobiLink Client Configuration to Use Transport Layer Security](#)

## 1.4 UltraLite Database Creation Approaches

There are three common types of database creation methods:

- Desktop creation methods with UltraLite administration tools designed for database creation.
- On-device creation methods with UltraLite APIs.  
On-device creation methods are primarily described in each API specific UltraLite programming book.

- A Central Administration remote task, configured to create an UltraLite database on a device.

Once the database is created, you can connect to it and build tables and other database objects.

#### In this section:

##### [Creating an UltraLite Database with the Create Database Wizard \[page 28\]](#)

Create a database using SQL Central.

##### [UltraLite Database Creation Using a Command Prompt \[page 29\]](#)

There are several utilities you can use to create a database at a command prompt.

##### [UltraLite Database Creation Using a MobiLink Synchronization Model \[page 29\]](#)

To simplify development, MobiLink includes a *Create Synchronization Model Wizard* to create your UltraLite database and server-side synchronization logic.

##### [UltraLite Database Creation Through Central Administration of Remote Databases \[page 30\]](#)

MobiLink provides a create database command that allows you to create an UltraLite database.

##### [Creating an UltraLite Database from an XML File \[page 30\]](#)

Use XML as an intermediate format for managing your UltraLite database.

##### [UltraLite Database Creation on a First Connection \[page 32\]](#)

You can program your application to create a new UltraLite database if one cannot be detected at connection time. The application can then use SQL to create tables, indexes, foreign keys, and so on. To populate the database, synchronize with a consolidated database.

##### [How to Access Creation Option Values \[page 32\]](#)

You cannot change creation option values after you have created a database. However, you can view the corresponding database properties in SQL Central.

##### [UltraLite Character Sets \[page 32\]](#)

The results of comparisons on strings, and the sort order of strings, in part depends on the character set, collation, and encoding properties of the database.

##### [Database Security \[page 35\]](#)

You can encrypt or obfuscate your databases. Encryption provides secure representation of the data in the database whereas obfuscation only prevents casual observation of the contents of the database.

## Related Information

[Accessing Database Options \[page 45\]](#)

## 1.4.1 Creating an UltraLite Database with the *Create Database Wizard*

Create a database using SQL Central.

### Context

Choose this method if you want help navigating the available database creation options. This wizard simplifies your choices by restricting what you can configure based on the target platform(s) you select. Once the database is created, it displays the command line syntax that you can record and use with the ulinit utility.

### Procedure

1. Click **Start** > **Programs** > **SQL Anywhere 17** > **Administration Tools** > **SQL Central**.
2. Click **Tools** > **UltraLite 17** > **Create Database**.
3. Follow the instructions in the *Create Database Wizard*.

### Results

The database is created.

### Next Steps

You can now connect to the database and build tables and other database objects.

### Related Information

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

## 1.4.2 UltraLite Database Creation Using a Command Prompt

There are several utilities you can use to create a database at a command prompt.

- Use the `ulinit` utility to create a new, empty UltraLite database or one sourced from a SQL Anywhere reference database schema. With this utility, you can include utility options to configure the database.
- Use the `uload` utility if you have an XML file that will serve as the source point for the schema and/or data of your new UltraLite database.
- Central Administration: Choose this method if you have a deployment where the MobiLink Agent is configured on all your deployed devices, or you are unable to deploy your initial UltraLite database with your application. You can configure a remote task to create a new UltraLite database on the device. This database can then be managed centrally by an administrator.

### Create a New UltraLite Database (Command Line)

Run the `ulinit` utility specifying the new UltraLite database file to accept the defaults:

```
ulinit test.udb
```

### Related Information

[Conversion from a SQL Anywhere Database to an UltraLite Database \[page 37\]](#)

[Manage Remote Databases](#)

[Creating an UltraLite Database from an XML File \[page 30\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

## 1.4.3 UltraLite Database Creation Using a MobiLink Synchronization Model

To simplify development, MobiLink includes a [Create Synchronization Model Wizard](#) to create your UltraLite database and server-side synchronization logic.

Choose this method if you are creating a synchronization system with remote UltraLite databases and a centralized consolidated database.

Once you have created your model, you can work in MobiLink Model mode in SQL Central to customize your synchronization model before you deploy it. When the model is ready, you can then deploy it to generate the scripts and tables required for your synchronization application.

## Related Information

[MobiLink Plug-in for SQL Central](#)

### 1.4.4 UltraLite Database Creation Through Central Administration of Remote Databases

MobiLink provides a create database command that allows you to create an UltraLite database.

## Related Information

[Manage Remote Databases](#)  
[Create Database Command](#)

### 1.4.5 Creating an UltraLite Database from an XML File

Use XML as an intermediate format for managing your UltraLite database.

## Prerequisites

UltraLite cannot use an arbitrary XML file. The `%SQLANY17%\Bin32` and `%SQLANY17%\Bin64` directories contains a `usm.xsd` file, containing the schema definition. Use this file to review the XML format.

## Context

You can use XML to:

- Load data into a new database with a different set of database properties/options.
- Upgrade the schema from a database created by a previous version of UltraLite.
- Create a text version of your UltraLite database.

## Procedure

1. Save the XML file to a directory of your choosing. You can either:

- Export/unload a database to an XML file. If you are unloading a SQL Anywhere database, use any of the supported export methods.
  - Take XML output from another source (that source could be another relational database or even a web site where transactions are recorded to a file). You must always ensure that the format of the XML meets the UltraLite requirements.
2. Run the ulload utility, including any necessary options.

## Results

The database is created.

## Example

To create a new UltraLite database in the file `sample.udb` from the table formats and data in `sample.xml`, run the following command:

```
ulload -c DBF=sample.udb sample.xml
```

## Next Steps

You can now connect to the database and build tables and other database objects.

## Related Information

[Relational Data Exported as XML](#)

[UltraLite Database Creation Using a Command Prompt \[page 29\]](#)

[Conversion from a SQL Anywhere Database to an UltraLite Database \[page 37\]](#)

[UltraLite Upgrades](#)

[Creating an UltraLite Database with the Create Database Wizard \[page 28\]](#)

[UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

## 1.4.6 UltraLite Database Creation on a First Connection

You can program your application to create a new UltraLite database if one cannot be detected at connection time. The application can then use SQL to create tables, indexes, foreign keys, and so on. To populate the database, synchronize with a consolidated database.

### Considerations

By adding the additional database creation and SQL code, your application size can grow considerably. However, this option can simplify deployment because you only need to deploy the application to the device. In some pre-production development cycles, you may want to delete and reconstruct the database on your device for testing purposes.

## 1.4.7 How to Access Creation Option Values

You cannot change creation option values after you have created a database. However, you can view the corresponding database properties in SQL Central.

For the UltraLiteJ API, you can use the `getCreationString` method to view the creation string registered with the `setCreationString` method.

For other UltraLite APIs, you can access the database properties programmatically from your UltraLite application by calling the `GetDatabaseProperty` function appropriate to the UltraLite API.

## 1.4.8 UltraLite Character Sets

The results of comparisons on strings, and the sort order of strings, in part depends on the character set, collation, and encoding properties of the database.

Choosing the correct character set, collation, and encoding properties for your database is primarily determined by:

- The desired sort order. Choose the collation that best sorts the characters you intend to store in your database.
- The platform of your device. Requirements among supported devices can vary, and some require that you use UTF-8 to encode your characters. If you need to support multiple devices, you need to determine whether a database can be shared.
- If you are synchronizing data, which languages and character sets are supported by the consolidated database. You must ensure that the character sets used in the UltraLite database and the consolidated database are compatible. Otherwise, data could be lost or become altered in unexpected ways if characters in one database's character set do not exist in the other's character set. If you have deployed UltraLite in a multilingual environment, you should also use UTF-8 to encode your UltraLite database.



When you synchronize, the MobiLink server converts characters as follows:

1. The UltraLite database characters are converted to Unicode.
2. The Unicode characters are converted into the consolidated database's character set.

**In this section:**

[UltraLite Platform Requirements for Character Set Encoding \[page 33\]](#)

Each platform has specific character set and encoding requirements.

[UltraLite Supported Collations \[page 34\]](#)

Several CHAR collations are supported in UltraLite.

## Related Information

[Character Sets](#)

[Character Set Considerations](#)

[Database Security \[page 35\]](#)

[UltraLite collation Creation Option \[page 150\]](#)

[UltraLite utf8\\_encoding Creation Option \[page 180\]](#)

[UltraLite Connection Parameters \[page 181\]](#)

[UltraLite case Creation Option \[page 147\]](#)

### 1.4.8.1 UltraLite Platform Requirements for Character Set Encoding

Each platform has specific character set and encoding requirements.

#### Microsoft Windows Desktop and Microsoft Windows Mobile

When using a UTF-8 encoded database on Microsoft Windows, you should pass wide characters to the database. If you use UTF-8 encoding on these platforms, UltraLite expects that non-wide string parameters are UTF-8 encoded, which is not a natural character set to use on Microsoft Windows. The exception is for connection strings, where string parameters are expected to be in the active code page. However, by using wide characters, you can avoid this complication.

## Related Information

[Character Sets](#)

[Character Set Considerations](#)

[Database Security \[page 35\]](#)

## 1.4.8.2 UltraLite Supported Collations

Several CHAR collations are supported in UltraLite.

You can also generate the list by executing the following command:

```
ulinit -Z
```

Collation label	Description
1250LATIN2	Code Page 1250, Windows Latin 2, Central/Eastern European
1250POL	Code Page 1250, Windows Latin 2, Polish
1251CYR	Code Page 1251, Windows Cyrillic
1252LATIN1	Code Page 1252, Windows Latin 1, Western
1252NOR	Code Page 1252, Windows Latin 1, Norwegian
1252SPA	Code Page 1252, Windows Latin 1, Spanish
1252SWEFIN	Code Page 1252, Windows Latin 1, Swedish/Finnish
1253ELL	Code Page 1253, Windows Greek, ISO8859-7 with extensions
1254TRK	Code Page 1254, Windows Turkish, ISO8859-9 with extensions
1254TRKALT	Code Page 1254, Windows Turkish, ISO8859-9 with extensions, I-dot e als I-no-dot
1255HEB	Code Page 1255, Windows Hebrew, ISO8859-8 with extensions
1256ARA	Code Page 1256, Windows Arabic, ISO8859-6 with extensions
1257LIT	Code Page 1257, Windows Baltic Rim, Lithuanian
874THAIBIN	Code Page 874, Windows Thai, ISO8859-11, binary ordering
932JPN	Code Page 932, Japanese Shift-JIS with Microsoft extensions
936ZHO	Code Page 936, Simplified Chinese, PRC GBK
949KOR	Code Page 949, Korean KS C 5601-1987 Encoding, Wansung
950ZHO_HK	Code Page 950, Traditional Chinese, Big 5 Encoding with HKSCS
950ZHO_TW	Code Page 950, Traditional Chinese, Big 5 Encoding
EUC_CHINA	Simplified Chinese, GB 2312-80 Encoding

Collation label	Description
EUC_JAPAN	Japanese EUC JIS X 0208-1990 and JIS X 0212-1990 Encoding
EUC_KOREA	Code Page 1361, Korean KS C 5601-1992 8-bit Encoding, Johab
EUC_TAIWAN	Code Page 964, EUC-TW Encoding
ISO1LATIN1	ISO8859-1, ISO Latin 1, Western, Latin 1 Ordering
ISO9LATIN1	ISO8859-15, ISO Latin 9, Western, Latin 1 Ordering
ISO_1	ISO8859-1, ISO Latin 1, Western
ISO_BINENG	Binary ordering, English ISO/ASCII 7-bit letter case mappings
UTF8BIN	UTF-8, 8-bit multibyte encoding for Unicode, binary ordering

## 1.4.9 Database Security

You can encrypt or obfuscate your databases. Encryption provides secure representation of the data in the database whereas obfuscation only prevents casual observation of the contents of the database.

By default, databases are not encrypted or obfuscated. Text and binary columns can be read when using a viewing tool such as a hex editor. Consider the following options if you do not want your data stored as plain text:

### Obfuscation

This option provides protection against casual attempts to access data in the database but does not provide as much security as strong encryption. Obfuscation has a minimal performance impact. You do not need any special configuration to use simple obfuscation on your device.

### AES 256-bit encryption

This option encrypts databases with an AES 256-bit algorithm. Strong encryption provides security against skilled and determined attempts to gain access to the data. You do not need any special configuration to use AES encryption on your device.

### FIPS 140-2 certified AES 256-bit encryption

Encryption libraries certified to comply to the FIPS 140-2 computer security standard *Security Requirements for Cryptographic Modules* are provided under a separate license. FIPS-certified AES encryption requires that you configure your device appropriately.

## Database obfuscation

To obfuscate data, specify `obfuscate=1` as a database creation parameter when you create your database. End users do not need to supply a corresponding connection parameter.

To obfuscate data with the UltraLiteJ API, use the `ConfigPersistent.enableObfuscation` method during database creation.

## Database encryption

Encryption keys should contain a combination of characters, numbers, and special symbols to be effective. Long encryption keys reduce the chances of someone guessing the key.

### **i** Note

After the database is encrypted, the encryption key cannot be recovered.

Using SQL Central wizards, you can specify UltraLite database encryption options during creation by clicking the *Encrypt the database* option and then clicking *Use strong encryption*. Select one of the AES algorithms and then enter an encryption key.

Using the ulinit utility, you can specify encryption using the -e option. Use the --fips option to specify whether to use FIPS-certified encryption. Specify the encryption key with the -k (--key) option.

UltraLite API encryption options are available when creating a database.

### **⚠** Caution

You can change the encryption key after the database has been created but only under extreme caution.

This operation is costly and is non-recoverable. You can lose your database entirely if your operation terminates mid-course.

For strongly encrypted databases, store a copy of the key in a safe location. If you lose the encryption key, there is no way to access the data, even with the assistance of Technical Support. The database must be discarded and you must create a new database.

The DBKEY parameter must be supplied when connecting to the database; otherwise, the connections fail. Encryption keys should be treated as sensitive information.

## Related Information

[Separately Licensed Components](#)

[Cache Size Adjustment for an UltraLite Database \[page 569\]](#)

[UltraLite obfuscate Creation Option \[page 165\]](#)

[UltraLite DBKEY Connection Parameter \[page 192\]](#)

[UltraLite fips Creation Option \[page 159\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

## 1.5 Conversion from a SQL Anywhere Database to an UltraLite Database

Create an UltraLite database from a SQL Anywhere reference database by running the ulinit utility with the `-a` option. The new database is created with the same settings as those in the reference database where possible.

The SQL Anywhere reference database acts as a database template, and uses the following settings to create an UltraLite database schema:

- Database configuration, such as the collation sequence
- Table definitions
- Synchronization publications

You can include data, and choose the columns, tables, and indexes as part of a publication in the reference database.

### i Note

To initialize an UltraLite database from an RDBMS other than SQL Anywhere, use the [Create Synchronization Model Wizard](#) in SQL Central, and connect to a consolidated database when prompted to obtain the schema information.

## Conversion Considerations

Prior to running the ulinit utility, consider if the following reference database tasks are required:

### **Add tables, keys, indexes, and synchronization publications as needed**

Add the tables and set primary keys as needed. You can also assign foreign key relationships that you need within your UltraLite application.

Indexes can improve performance dramatically, particularly on slow devices. Primary key columns are automatically indexed, but other types of columns are not.

### i Note

If your UltraLite application frequently retrieves information in a particular order, consider adding an index to your reference database specifically for this purpose.

Use synchronization publications to synchronize different tables at different times. You can use multiple synchronization publications to define table subsets and set the synchronization priority with them.

### **Update database options or table schema that may have undesired effects**

For example, if a column in the SQL Anywhere database includes a clause that UltraLite does not support, the default value is ignored and the UltraLite default is specified for the new database.

### **Change the collation sequence if it is not supported by UltraLite**

UltraLite uses the name of the collation sequence that is defined in the reference database. You can still choose to use UTF-8 to encode the database by setting the `utf8_encoding` property.

To see a list of collations and corresponding codepages supported by UltraLite, run `ulinit` with the `-Z` option at a command prompt. If the reference database uses a collation sequence that is not supported, such as UCA for CHAR collation sequences, change the collation sequence to one that is supported by performing the following steps:

1. Use the Unload utility to unload the SQL Anywhere reference database.
2. Create a new SQL Anywhere database with a different collation and run the `reload.sql` script through Interactive SQL.

## Example

The following command creates a new UltraLite database named `customer.udb` from an existing SQL Anywhere reference database defined in the `MySADb` data source. Tables in the reference database are defined in `TestPublication`. The created UltraLite database contains all the same database options and tables contained in `TestPublication`, and is encrypted with the `mykey` encryption key.

```
ulinit -a "DSN=MySADb;UID=JimmyB;PWD=secret" -n TestPublication -k mykey customer.udb
```

The following command creates a new UltraLite database named `customer.udb` from an existing SQL Anywhere database named `MySource.db`. The tables and indexes in the created database match those contained in the `Pub1` schema publication. The `Pub2` synchronization publication is created in the UltraLite database.

```
ulinit -a DBF=MySource.db;UID=JimmyB;PWD=secret customer.udb -n Pub1 -s Pub2
```

## Related Information

[UltraLite Tables and Columns \[page 53\]](#)

[When to Use an Index \[page 63\]](#)

[Index Scan Creation and Maintenance \[page 570\]](#)

[UltraLite Database Creation Approaches \[page 26\]](#)

[Publishing Data in UltraLite \[page 83\]](#)

[Unload Utility \(dbunload\)](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

## 1.6 UltraLite Database Connections

Applications that use a database must establish a connection to that database before transactions can occur. By connecting to an UltraLite database, you form a channel through which all activity from the application takes place.

An application can be an UltraLite utility, a connection window, or your own custom application. Each connection attempt creates a database specific SQL transaction.

### In this section:

#### [UltraLite Connection Strings and Parameters \[page 39\]](#)

A **connection string** is a set of connection parameters that is passed from an application to the UltraLite runtime so that a database connection can be defined and established. Some parameters are always required to open a connection while others are used to adjust database features for a single connection.

#### [UltraLite Connection Parameters and the ULSQLCONNECT Environment Variable \[page 41\]](#)

Use the ULSQLCONNECT environment variable to avoid having to supply the same connection parameters repeatedly to the UltraLite desktop administration tools. Both Interactive SQL and SQL Central support the ULSQLCONNECT environment variable.

#### [UltraLite File Path Formats in Connection Parameters \[page 42\]](#)

The physical storage of your device determines whether the database is saved as a file and what naming conventions you must follow when identifying your database.

## Related Information

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite Connection Parameters \[page 181\]](#)

### 1.6.1 UltraLite Connection Strings and Parameters

A **connection string** is a set of connection parameters that is passed from an application to the UltraLite runtime so that a database connection can be defined and established. Some parameters are always required to open a connection while others are used to adjust database features for a single connection.

Connection strings are defined as `keyword=value` pairs in a semicolon-delimited list. The following example illustrates a connection string fragment that specifies a database file name, user ID, and password:

```
DBF=myULdb.udb;UID=JDoe;PWD=token
```

Methods of supplying these parameters to a database can vary depending on whether you are connecting from an UltraLite utility or an UltraLite application.

UltraLite command line utilities typically use a connection string if a connection to a database is required.

UltraLite applications can be developed using an UltraLite API to read connection parameter values from a stored file or in the application code. You can supply fixed connection strings when user authentication is not required, or prompt users to supply parameter values at connection time.

When a connection string has been assembled, it is passed to the UltraLite runtime for processing. The connection to the database is granted when the connection attempt is validated. Connection failures can occur if the database file does not exist, or the authentication was not successful.

UltraLite generates an error when it encounters an unrecognized connection parameter.

## Prefixes

You can use a prefix with a connection parameter to specify that a parameter applies only when an application is running on a particular output. For UltraLite, these prefixes are `desktop:` and `device:`.

## Restrictions

Any leading and/or trailing spaces in connection string parameter values are ignored. Connection parameter values cannot include leading single quotes (`'`), leading double quotes (`"`), or semicolons (`:`).

## Example

For example, if you use the `ulload` utility, the following connection string is used to load new XML data into an existing database. You cannot connect to the database file until you supply this string:

```
ulload -c "DBF=sample.udb;UID=DBA;PWD=sql" sample.xml
```

### In this section:

[Precedence of Connection Parameters for UltraLite Administration Tools \[page 41\]](#)

UltraLite administration tools follow a specific order of connection parameter precedence:

## Related Information

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite Connection Parameters \[page 181\]](#)



## 1.6.1.1 Precedence of Connection Parameters for UltraLite Administration Tools

UltraLite administration tools follow a specific order of connection parameter precedence:

- Device-specific options take precedence over nonspecific options. For example: `device:DBF` takes precedence over `DBF` on Microsoft Windows Mobile devices.
- Desktop-specific options take precedence over nonspecific options. For example: `desktop:DBF` takes precedence over `DBF` on Microsoft Windows.
- If you supply duplicate parameters in a connection string, the last one supplied is used. All others are ignored.
- Parameters in the connection string take precedence over those supplied in the `ULSQLCONNECT` environment variable or a connection object.
- If no value is supplied for *both* `UID` and `PWD` in either the connection string or `ULSQLCONNECT`, the defaults of `UID=DBA` and `PWD=sql` are assumed.

## 1.6.2 UltraLite Connection Parameters and the ULSQLCONNECT Environment Variable

Use the `ULSQLCONNECT` environment variable to avoid having to supply the same connection parameters repeatedly to the UltraLite desktop administration tools. Both Interactive SQL and SQL Central support the `ULSQLCONNECT` environment variable.

You can set the `ULSQLCONNECT` environment variable to contain a list of parameters defined as `keyword=value` pairs in a semicolon-delimited list.

The supplied values become defaults for the desktop administration tools. If any additional parameters are required or if you must override default values set with this environment variable, ensure that you set these values. User-supplied values always take precedence over this environment variable.

### Example

In this example, you set the `ULSQLCONNECT` environment variable to specify a default database file named `c:\database\myfile.udb`, a default user `demo`, and a default password `test`.

```
set ULSQLCONNECT=DBF=c:\database\myfile.udb;UID=demo;PWD=test
```

If you launch Interactive SQL with no arguments, then you will automatically connect to the specified database using the specified credentials. If you launch SQL Central and connect to an UltraLite database, these parameters are used to populate the connection dialog.

## Related Information

[Precedence of Connection Parameters for UltraLite Administration Tools \[page 41\]](#)

### 1.6.3 UltraLite File Path Formats in Connection Parameters

The physical storage of your device determines whether the database is saved as a file and what naming conventions you must follow when identifying your database.

#### i Note

Use absolute file paths when using the UltraLite engine to support multi-process access to a database since the engine may be started in different locations.

The DBF parameter is most appropriate when targeting a single deployment platform or when using UltraLite desktop administration tools. For example:

```
ulload -c DBF=sample.udb sample.xml
```

#### i Note

You can use the UltraLite administration tools to administer databases already deployed to an attached device.

Otherwise, if you are writing a cross-platform application, use the platform-specific DBF connection parameters to construct a universal connection string. For example:

```
Connection = DatabaseManager::OpenConnection("UID=JDoe;PWD=ULdb;device:DBF=\database\MobileDB.udb;desktop:DBF=DesktopDB.udb")
```

## Platforms Other Than Microsoft Windows Mobile

Either absolute or relative paths are allowed.

### Microsoft Windows Mobile

Microsoft Windows Mobile devices require absolute paths.

You can administer a Microsoft Windows Mobile database on either the desktop or the attached device. To administer a database on a Microsoft Windows Mobile device, ensure you prefix the absolute path with [wce:\](#). For example, using the ulunload utility:

```
ulunload -c DBF=wce:\UltraLite\myULdb.udb c:\out\ce.xml
```

In this example, UltraLite unloads the database from the Microsoft Windows Mobile device to the `ce.xml` file in the Microsoft Windows desktop folder of `c:\out`.

## Related Information

[UltraLite DBF Connection Parameter \[page 191\]](#)

# 1.7 UltraLite Database Tasks and Features

There are many tasks you perform and features you can use to manage UltraLite databases.

### In this section:

[Reading Database Properties \[page 43\]](#)

Inspect the settings of any UltraLite database property.

[Accessing Database Options \[page 45\]](#)

View and change database options to configure database behavior.

[UltraLite Event Notifications \[page 46\]](#)

UltraLite supports events and notifications. A notification is a message that is sent when an event occurs, providing additional parameter information.

[Isolation Levels \[page 48\]](#)

Isolation levels define the degree to which the operations in one transaction are visible to the operations in other concurrent transactions.

[Validating an UltraLite Database \[page 51\]](#)

Validate your database using tools such as the *Validate Database Wizard* in SQL Central, the UltraLite Validate Database utility, or the `ValidateDatabase` method in your UltraLite API.

[UltraLite Database Back up and Recovery \[page 52\]](#)

If an application using an UltraLite database stops unexpectedly, the database automatically recovers to a consistent state when the application is restarted.

## 1.7.1 Reading Database Properties

Inspect the settings of any UltraLite database property.

### Context

Each UltraLite API contains a `GetDatabaseProperty` method that you can use in your applications to access database properties.

You can also access database properties by calling the DB\_PROPERTY SQL function.

## Procedure

1. Using SQL Central, connect to the database.
2. Right-click the database and click *Properties*.

## Results

In the *Database Properties* window, database properties are listed on the *General* and *Synchronization Information* tabs. On the *Synchronization Information* tab, the database properties are listed alphabetically by the property name. To sort database properties by the value, click the *Value* column.

## Related Information

[UltraLite Database Properties \[page 203\]](#)  
[DB\\_PROPERTY Function \[System\] \[page 390\]](#)  
[How to Access Creation Option Values \[page 32\]](#)  
[UltraLite Character Sets \[page 32\]](#)  
[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)  
[UltraLite Load XML to Database Utility \(uload\) \[page 234\]](#)  
[UltraLite DBF Connection Parameter \[page 191\]](#)  
[UltraLite DBKEY Connection Parameter \[page 192\]](#)  
[UltraLite Desktop Connection Parameter Prefix \[page 194\]](#)  
[UltraLite Device Connection Parameter Prefix \[page 195\]](#)  
[UltraLite MIRROR\\_FILE Connection Parameter \[page 196\]](#)  
[UltraLite PWD Connection Parameter \[page 198\]](#)  
[UltraLite RESERVE\\_SIZE Connection Parameter \[page 199\]](#)  
[UltraLite UID Connection Parameter \[page 202\]](#)

## 1.7.2 Accessing Database Options

View and change database options to configure database behavior.

### Context

You can view and change the setting of persistent database options from SQL Central. Temporary UltraLite database options *cannot* be viewed or set from SQL Central.

Database options can be set or modified at any time. Temporary database options only persist while the database is running.

Option values are set by using the SET OPTION SQL statement.

### Procedure

1. Using SQL Central, connect to the database.
2. Right-click the database and click *Options*.
3. To set or reset an option, type a new value in the *Value* field.
4. Click *Set Now* or *Reset Now* to commit the change.

### Results

The database option setting is changed and saved.

### Related Information

[UltraLite Database Options \[page 206\]](#)

[Reading Database Properties \[page 43\]](#)

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite Character Sets \[page 32\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

[UltraLite DBF Connection Parameter \[page 191\]](#)

[UltraLite DBKEY Connection Parameter \[page 192\]](#)

[UltraLite Desktop Connection Parameter Prefix \[page 194\]](#)

[UltraLite Device Connection Parameter Prefix \[page 195\]](#)

[UltraLite MIRROR\\_FILE Connection Parameter \[page 196\]](#)

[UltraLite PWD Connection Parameter \[page 198\]](#)

[UltraLite RESERVE\\_SIZE Connection Parameter \[page 199\]](#)

[UltraLite UID Connection Parameter \[page 202\]](#)

## 1.7.3 UltraLite Event Notifications

UltraLite supports events and notifications. A notification is a message that is sent when an event occurs, providing additional parameter information.

Event notifications allow you to provide coordination and signaling between connections or applications connected to the same database. Notifications are managed in queues: either a connection's default queue or, optionally, queues that are explicitly created and named. When an event occurs, notifications are sent to registered queues (or connections).

Each connection manages its own notification queues. Named queues can be created for any connection.

Using predefined system events this feature also provides "triggers" for changes to data, such as when a change is made to a table, for example, or signaling when a synchronization has occurred. Predefined events include:

- Commit
- SyncComplete
- TableModified

UltraLite has system events and events can also be user-defined. User events may also be defined and triggered by an application.

APIs for events and notifications are provided in each supported language. Additionally, a SQL function is provided to access the API functionality.

### Events

Event	Occurrence
<i>Commit</i>	Signaled upon completion of a commit.
<i>SyncComplete</i>	Signaled upon completion of a sync.

## Event

### *TableModified*

## Occurrence

Triggered when rows in a table are inserted, updated, or deleted. One event is signaled per request, no matter how many rows were affected by the request when registering for the event.

The `object_name` parameter specifies the table to monitor. A value of "\*" means all tables in the database.

The `table_name` notification parameter is the name of the modified table.

```
note_info.event_name = "SyncComplete";
note_info.event_name_len = 12;
note_info.parms_type = ul_ev_note_info::P_NONE;
```

```
note_info.event_name = "TableModified";
note_info.event_name_len = 13;
note_info.parms_type = ul_ev_note_info::P_TABLE_NAME;
note_info.parms = table->name->data;
note_info.parms_len = table->name->len;
```

## Working with Queues

Queues can be created and destroyed.

*CreateNotificationQueue* creates an event notification queue for the current connection. Queue names are scoped per-connection, so different connections can create queues with the same name. When an event notification is sent, all queues in the database with a matching name receive a separate instance of the notification. Names are case insensitive. A default queue is created on demand for each connection if no queue is specified. This call fails with an error if the name already exists for the connection or isn't valid.

*DestroyNotificationQueue* destroys the given event notification queue. A warning is signaled if unread notifications remain in the queue. Unread notifications are discarded. A connection's default event queue, if created, is destroyed when the connection is closed.

## Working with events

*DeclareEvent* declares an event which can then be registered for and triggered. UltraLite predefines some system events triggered by operations on the database or the environment. The event name must be unique and names are case insensitive. Returns true if the event was declared successfully, false if the name is already used or invalid.

*RegisterForEvent* registers a queue to receive notifications of an event. If no queue name is supplied, the default connection queue is implied, and created if required. Certain system events allow specification of an object name to which the event applies. For example, the TableModified event can specify the table name. Unlike SendNotification, only the specific queue registered will receive notifications of the event; other queues with

the same name on different connections will not (unless they are also explicitly registered). Returns true if the registration succeeded, false if the queue or event does not exist.

*TriggerEvent* triggers an event and sends a notification to all registered queues. Returns the number of event notifications sent. Parameters may be supplied as name=value; pairs.

## Working with Notifications

*SendNotification* sends a notification to all queues in the database matching the given name (including any such queue on the current connection). This call does not block. Use the special queue name "\*" to send to all queues. Returns the number of notifications sent (the number of matching queues). Parameters may be supplied as name=value; pairs.

*GetNotification* reads an event notification. This call blocks until a notification is received or until the given wait period expires. To cancel a wait, send another notification to the given queue or use *CancelGetNotification*. After reading a notification, use *ReadNotificationParameter* to retrieve additional parameters. Returns true if an event was read, false if the wait period expired or was canceled.

*GetNotificationParameter* gets a named parameter for the event notification just read by *GetNotification*. Only the parameters from the most-recently read notification on the given queue are available. Returns true if the parameter was found, false if the parameter was not found.

*CancelGetNotification* cancels any pending *GetNotification* calls on all queues matching the given name. Returns the number of affected queues (not necessarily the number of blocked reads).

## Other Considerations

- Notification queue and event names are limited to 32 characters.
- To govern system resources, the number of notifications is limited. When this limit is exceeded, `SQL_EVENT_NOTIFICATION_QUEUE_FULL` is signaled and the pending notification is discarded.

## 1.7.4 Isolation Levels

Isolation levels define the degree to which the operations in one transaction are visible to the operations in other concurrent transactions.

UltraLite uses the default isolation level, `read_committed`, for connections. The default UltraLite isolation level aids data consistency by isolating uncommitted rows.



Isolation level	Characteristics
0 - read_uncommitted	<ul style="list-style-type: none"> <li>Allow dirty reads, non-repeatable reads, and phantom rows.</li> <li>No guarantee that concurrent transactions will not modify row or roll back changes to rows</li> </ul>
1 - read_committed	<ul style="list-style-type: none"> <li>Allow non-repeatable reads and phantom rows</li> <li>Prevent dirty reads</li> <li>No guarantee that query results will not change during transactions</li> </ul>

You can change the isolation level from read\_committed to read\_uncommitted using one of following approaches:

- Use the SET OPTION SQL statement and the isolation\_level database option. For example, the following statement sets the isolation level to read uncommitted:

```
SET OPTION isolation_level = 'READ_UNCOMMITTED'
```

- For the UltraLite C++ API, use the ULConnection.SetDatabaseOption method to change the isolation level. For the UltraLite.NET API, use the ULConnection.BeginTransaction or ULDatabaseSchema.SetDatabaseOption methods to create a transaction with the read\_committed isolation level. For the UltraLiteJ API, use the Connection.setOption method.

#### In this section:

[Characteristics of the read\\_uncommitted Isolation Level \[page 49\]](#)

Several side effects are possible when UltraLite operates at an isolation\_level of 0 (read\_uncommitted).

## Related Information

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

[UltraLite isolation\\_level Option \[page 211\]](#)

### 1.7.4.1 Characteristics of the read\_uncommitted Isolation Level

Several side effects are possible when UltraLite operates at an isolation\_level of 0 (read\_uncommitted).

- Applications can read uncommitted data (dirty reads). In this scenario, transactions may access rows in the database that are not committed and may still get rolled back by another transaction. This phenomena can result in phantom rows (rows that get added after the original query, making the result set returned in a repeated, duplicate query different).

- Applications can perform non-repeatable reads. In this scenario, an application reads a row from the database, and then goes on to perform other operations. Then a second application updates/deletes the row and commits the change. If the first application attempts to re-read the original row, it receives either the updated information or discovers that the original row was deleted.

## Example

Consider two connections, A and B, each with their own transactions.

1. As connection A works with the result set of a query, UltraLite **fetches** a copy of the current row into a buffer.

### i Note

Reading or fetching a row does not lock the row. If connection A fetches but does not modify a row, connection B can still modify the row.

2. As A modifies the current row, it changes the copy in the buffer. The copy in the buffer is written back into the database when connection A calls an Update method or closes the result set.
3. A write lock is placed on the row to prevent other transactions from modifying it. This modification is uncommitted, until connection A performs a commit.
4. Depending on the modification, if connection B fetches the current row, it may experience the following:

Connection A's modification	Result <sup>1</sup>
Row has been deleted.	Connection B gets the next row in the result set.
Row has been modified.	Connection B gets the latest copy of the row.

<sup>1</sup> Queries used by Connection A and B do not contain temporary tables. Temporary tables can cause other side effects.

## Related Information

[Tutorial: Understanding Dirty Reads](#)

[Tutorial: Understanding Phantom Rows](#)

[Tutorial: Understanding Non-repeatable Reads](#)

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

[UltraLite isolation\\_level Option \[page 211\]](#)

## 1.7.5 Validating an UltraLite Database

Validate your database using tools such as the [Validate Database Wizard](#) in SQL Central, the UltraLite Validate Database utility, or the `ValidateDatabase` method in your UltraLite API.

### Context

Database file corruption may not be reported until the database server tries to access the affected part of the database.

#### ⚠ Caution

Database validation should be performed while no connections are making changes to the database; otherwise, errors indicating database corruption might be reported even though no corruption actually exists.

You can validate an UltraLite database using any of the following methods:

- The [Validate Database Wizard](#) in SQL Central.
- The `ulvalid` command line utility.
- The `ValidateDatabase` method in your UltraLite API.

### Procedure

1. In the left pane of SQL Central, click the UltraLite database.
2. Click **File** > [Validate Database](#).
3. Follow the instructions in the [Validate Database Wizard](#).

### Results

The database is validated.

### Related Information

[UltraLite Validate Database Utility \(ulvalid\) \[page 246\]](#)

## 1.7.6 UltraLite Database Back up and Recovery

If an application using an UltraLite database stops unexpectedly, the database automatically recovers to a consistent state when the application is restarted.

All committed transactions flushed to memory before the unexpected failure are present in the database. All transactions not flushed at the time of the failure are rolled back.

An UltraLite database does not use a transaction log to perform recovery. Instead, UltraLite stores state information for every row to determine the fate of a row when recovering.

### Backups

UltraLite provides protection against system failures, but not from media failures. The best way to make a backup of an UltraLite application is to synchronize with a consolidated database. To restore an UltraLite database, start with an empty database and populate it from the consolidated database through synchronization.

### Related Information

[UltraLite Database Row State Management \[page 584\]](#)

[Flush Single or Grouped Transactions \[page 586\]](#)

[UltraLite as a MobiLink Client \[page 72\]](#)

## 1.8 UltraLite Database Schemas

The logical framework of the database is known as a **schema**.

### UltraLite Database Schemas

You can upgrade the schema of an UltraLite database with the appropriate Data Definition Language (DDL) statements or by using the ALTER DATABASE SCHEMA FROM FILE statement to modify the schema definition using a SQL script.

Schema changes can take a considerable amount of time. For example, all rows in the associated table must be updated when the column type is changed. DDL statements successfully execute when there are not any:

- Uncommitted transactions.
- Other active uses of the database, such as synchronization, prepared but unreleased statements, or executing database operations.

When the DDL statement is executing, any other attempt to use the database is blocked until the DDL statement completes the schema change.

**In this section:**

[UltraLite Tables and Columns \[page 53\]](#)

Tables are used to store data and define the relationships for data in them. Tables consist of rows and columns. Each column carries a particular kind of information, such as a phone number or a name, while each row specifies a particular entry.

[UltraLite Indexes \[page 62\]](#)

An index is a set of pointers to rows in a table based on the order of the values of data in one or more table columns.

[UltraLite Users \[page 66\]](#)

A typical UltraLite database contains one user ID and password. UltraLite databases are created with a default user ID of *DBA* and default password of *sql* unless otherwise specified.

## Related Information

[Deploying UltraLite Database Schema Upgrades \[page 131\]](#)

[ALTER DATABASE SCHEMA FROM FILE Statement \[UltraLite\] \[page 520\]](#)

## 1.8.1 UltraLite Tables and Columns

Tables are used to store data and define the relationships for data in them. Tables consist of rows and columns. Each column carries a particular kind of information, such as a phone number or a name, while each row specifies a particular entry.

When you first create an UltraLite database, the only tables you will see are the system tables. System tables hold the UltraLite schema. You can hide or show these tables from SQL Central as needed.

You can then add new tables as required by your application. You can also browse data in those tables, and copy and paste data among existing tables in the source database or even among other open destination databases.

In UltraLite, you can only create base tables, which you declare to hold persistent data. UltraLite does not support global temporary or declared temporary tables.

**In this section:**

[Row Packing and Table Definitions \[page 54\]](#)

UltraLite works with rows in two formats: unpacked rows and packed rows.

[Creating UltraLite Tables \[page 55\]](#)

Create base tables to hold your persistent relational data.

[Adding a Column to an UltraLite Table \[page 56\]](#)

Add column to an UltraLite table after it has been created.

### [Altering UltraLite Column Definitions \[page 57\]](#)

Change the structure of column definitions for a table by altering various column attributes or deleting columns entirely.

### [Deleting UltraLite Tables \[page 58\]](#)

Delete tables when you no longer need them.

### [Browsing the Information in UltraLite Tables \[page 59\]](#)

View the data held within the tables of an UltraLite database.

### [Data Copying and Pasting to or from UltraLite Databases \[page 60\]](#)

With SQL Central you can copy and paste and drag and drop. This data transferral allows you to share or move objects among one or more databases. By copying and pasting or dragging and dropping you can share data described by the table that follows.

### [Viewing Entity-relationship \(ER\) Diagrams in SQL Central \[page 61\]](#)

View and configure an entity-relationship (ER) diagram of the tables in an UltraLite database by using SQL Central.

## Related Information

[Database Creation](#)

[UltraLite System Tables \[page 248\]](#)

### 1.8.1.1 Row Packing and Table Definitions

UltraLite works with rows in two formats: unpacked rows and packed rows.

#### **Unpacked rows**

are the uncompressed format. Each row must be unpacked before individual column values can be read or written.

#### **Packed rows**

are the compressed representation of the unpacked row, where each of the column values is compressed so that the entire row takes up as little memory as possible. The size of a packed row depends on the values in each column: for example, two rows can belong to the same table, but can differ significantly in their packed size. Note also that LONG BINARY and LONG VARCHAR columns are stored separate from the packed row.

UltraLite has a limitation that a packed row must fit on a database page. Since LONG BINARY and LONG VARCHAR columns are not stored with the packed row, they can exceed the page size.

It is important to understand that table definitions describe the row *before* the UltraLite runtime packs the data. Because the size of a packed row depends on the values in each column, you cannot readily pre-determine from the table definition whether the packed row requirement is satisfied. For this reason, UltraLite allows you to define a table where an unpacked row would not fit on a page. To know if a row fits on a page, you must try inserting or updating the row itself; if a row does not fit, UltraLite detects and reports this error.

## i Note

You cannot declare tables to be any large size. UltraLite maintains a declared table row size limit of 64 KB. If you try to define a table where an unpacked row can exceed this maximum, UltraLite generates a SQL error code of `SQL_MAX_ROW_SIZE_EXCEEDED (-1132)`.

## Related Information

[Database Creation](#)

[UltraLite System Tables \[page 248\]](#)

[UltraLite `page\_size` Creation Option \[page 166\]](#)

## 1.8.1.2 Creating UltraLite Tables

Create base tables to hold your persistent relational data.

### Prerequisites

Tables in UltraLite applications must include a primary key. Primary keys are also required during MobiLink synchronization, to associate rows in the UltraLite database with rows in the consolidated database.

### Procedure

1. Using SQL Central, connect to the UltraLite database.
2. In the left pane, right-click *Tables* and click **New > Table**.
3. In the *What Do You Want To Name The New Table* field, type the new table name.
4. Click *Finish*.
5. From the *File* menu, click *Save*.

### Results

The table is created. The table and any data it contains exist until you explicitly delete the data or drop the table.

## Next Steps

Add columns or create indexes.

## Related Information

[Adding a Column to an UltraLite Table \[page 56\]](#)

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

### 1.8.1.3 Adding a Column to an UltraLite Table

Add column to an UltraLite table after it has been created.

## Prerequisites

If the table already holds data, you can only add a column if the column definition includes a default value or allows NULL values.

## Procedure

1. Using SQL Central, connect to the UltraLite database.
2. In the left pane, double-click *Tables*.
3. Double-click a table.
4. Click the *Columns* tab, right-click the white space below the table and click ► *New* ► *Column* ▾.
5. Set the attributes for the new column and then save your changes.

## Results

The column is added to the table.

## Related Information

[SQL Data Types \[page 288\]](#)



[Column Data Type Considerations](#)

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

[ALTER TABLE Statement \[UltraLite\] \[page 524\]](#)

## 1.8.1.4 Altering UltraLite Column Definitions

Change the structure of column definitions for a table by altering various column attributes or deleting columns entirely.

### Prerequisites

The modified column definition must suit the requirements of any data already stored in the column. For example, you cannot alter a column to disallow NULL if the column already has a NULL entry.

### Procedure

1. Using SQL Central, connect to the UltraLite database.
2. In the left pane, double-click *Tables*.
3. Double-click a table.
4. Click the *Columns* tab and alter the column attributes.
5. From the *File* menu, click *Save Table*.

### Results

The table is saved with the new column attributes.

### Related Information

[SQL Data Types \[page 288\]](#)

[Column Data Type Considerations](#)

[ALTER TABLE Statement \[UltraLite\] \[page 524\]](#)

## 1.8.1.5 Deleting UltraLite Tables

Delete tables when you no longer need them.

### Prerequisites

You can drop any table if the table:

- Is not being used as an article in a publication.
- Does not have any columns that are referenced by another table's foreign key.

In these cases, you must change the publication or delete the foreign key *before* you can successfully delete the table.

### Procedure

1. Using SQL Central, connect to the UltraLite database.
2. In the left pane, double-click [Tables](#).
3. Right-click a table and click [Delete](#).
4. Click [Yes](#).

### Results

The table is deleted.

### Related Information

[DROP TABLE Statement \[UltraLite\] \[page 548\]](#)

## 1.8.1.6 Browsing the Information in UltraLite Tables

View the data held within the tables of an UltraLite database.

### Prerequisites

The database must be connected and selected.

### Context

Tables can be user tables or system tables. You can filter tables by showing and hiding system tables from your current view of the database. Because UltraLite does not have a concept of ownership, all users can browse all tables.

### Procedure

1. Using SQL Central, connect to the UltraLite database.
2. To view a list of tables, double-click *Tables*.
3. To view table data, double-click a table and click the *Data* tab in the right pane.

### Results

The tables and data are displayed.

### Related Information

[UltraLite System Tables \[page 248\]](#)

## 1.8.1.7 Data Copying and Pasting to or from UltraLite Databases

With SQL Central you can copy and paste and drag and drop. This data transferral allows you to share or move objects among one or more databases. By copying and pasting or dragging and dropping you can share data described by the table that follows.

Target	Result
Another UltraLite or SQL Anywhere database.	A new object is created, and the original object's code is copied to the new object.
The same UltraLite database.	A copy of the object is created; you must rename the new object.

### i Note

You can copy data from a database opened in MobiLink and paste it into an UltraLite database. However, you cannot paste UltraLite data into a database opened in MobiLink.

## SQL Central

When you copy any of the following objects in the UltraLite plug-in, the SQL for the object is also copied to the clipboard. You can paste this SQL into other applications, such as Interactive SQL or a text editor. For example, if you copy an index in SQL Central and paste it into a text editor, the CREATE INDEX statement for that index appears. You can copy the following objects in the UltraLite plug-in:

- Articles
- Columns
- Foreign keys
- Indexes
- Publications
- Tables
- Unique constraints

## Interactive SQL

With Interactive SQL you can also copy data from a result set into another object.

- Use the SELECT statement results into a named object.
- Use the INSERT statement to insert a row or selection of rows from elsewhere in the database into a table.

## Related Information

[INSERT Statement \[UltraLite\] \[page 552\]](#)

[SELECT Statement \[UltraLite\] \[page 558\]](#)


### 1.8.1.8 Viewing Entity-relationship (ER) Diagrams in SQL Central

View and configure an entity-relationship (ER) diagram of the tables in an UltraLite database by using SQL Central.

#### Context

UltraLite does not allow users to configure the ER diagram by importing or exporting a layout or by filtering the tables that appear by owner.

#### Procedure

1. In SQL Central, use the *UltraLite 17* plug-in to connect to a database.
2. Select the database, and then click the *ER Diagram* tab in the right pane to see the diagram.
3. Click **File** > *Choose ER Diagram Tables* , and specify the tables to appear in the ER diagram.
4. Click *OK*.
5. Arrange the objects in the diagram as needed.  
Drag the objects to change the layout.
6. (Optional) Double-click a table to see the column definitions for that table.

#### Results

The entity-relationship diagram appears in SQL Central.

## Related Information

[Database Creation](#)

[Foreign Keys](#)

## 1.8.2 UltraLite Indexes

An index is a set of pointers to rows in a table based on the order of the values of data in one or more table columns.

The index is a database object that is maintained automatically by UltraLite after it has been created. When UltraLite optimizes a query, it scans existing indexes to see if one exists for the table(s) named in the query. If it can help UltraLite return rows more quickly, the index is used. If you are using the UltraLite Table API in your application, you can specify an index that helps determine the order in which rows are traversed.

### i Note

Indexes can improve the performance of a query, especially for large tables. To see whether a query is using a particular index, you can check the execution plan with Interactive SQL.

Alternatively, your UltraLite applications can include PreparedStatement objects which have a method to return plans.

UltraLite supports the following indexes. These indexes can be single or multi-column (also known as composite indexes). You cannot index LONG VARCHAR or LONG BINARY columns.

Index	Characteristics
Primary key	Required. An instance of a unique key. You can only have one primary key. Values in the indexed column or columns must be unique and cannot be NULL.
Foreign key <sup>1</sup>	Optional. Values in the indexed column or columns can be duplicated. Nullability depends on whether the column was created to allow NULL. Values in the foreign key columns must exist in the table being referenced
Unique key <sup>2</sup>	Optional. Values in the indexed column or columns must be unique and cannot be NULL.
Non-unique index	Optional. Values in the indexed column or columns can be duplicated and can be NULL.
Unique index	Optional. Values in the indexed column or columns cannot be duplicated and can be NULL.

<sup>1</sup> A foreign key can reference either a primary key or a unique key.

<sup>2</sup> Also known as a unique constraint.

## About Composite Indexes

Multi-column indexes are sometimes called composite indexes. Additional columns in an index can allow you to narrow down your search, but having a two-column index is not the same as having two separate indexes. For example, the following statement creates a two-column composite index:

```
CREATE INDEX name
ON Employees ( Surname, GivenName )
```

A composite index is useful if the first column alone does not provide high selectivity. For example, a composite index on Surname and GivenName is useful when many employees have the same surname. A composite index on EmployeeID and Surname would not be useful because each employee has a unique ID, so the column Surname does not provide any additional selectivity.

#### In this section:

[When to Use an Index \[page 63\]](#)

Indexes improve performance when querying data.

[Index Types \[page 64\]](#)

UltraLite supports different types of indexes: unique keys, unique indexes, and non-unique indexes. What differentiates one from the others is what is allowed in that index.

[Adding an UltraLite Index \[page 65\]](#)

Adding indexes to databases speeds up the search process.

[Dropping an UltraLite Index \[page 66\]](#)

Drop an index from the database.

## Related Information

[Index Scan Creation and Maintenance \[page 570\]](#)

[Execution Plans in UltraLite \[page 575\]](#)

[Composite Indexes](#)

[Data creation and modification in UltraLite.NET using the ULTable Class \[page 610\]](#)

[Data Creation and Modification in UltraLite C++ Using the ULTable Class \[page 657\]](#)

### 1.8.2.1 When to Use an Index

Indexes improve performance when querying data.

Use an index when:

#### **You want UltraLite to maintain referential integrity**

An index also affords UltraLite a means of enforcing a uniqueness constraint on the rows in a table. You do not need to add an index for data that is very similar.

#### **The performance of a particular query is important to your application**

If an index improves performance of a query and the performance of that query is important to your application and is used frequently, then you want to maintain that index. Unless the table in question is extremely small, indexes can improve search performance dramatically. Indexes are typically recommended whenever you search data frequently.

#### **You have complicated queries**

More complicated queries, (for example, those with JOIN, GROUP BY, and ORDER BY clauses), can yield substantial improvements when an index is used, though it may be harder to determine the degree to

which performance has been enhanced. Therefore, test your queries both with and without indexes, to see which yields better performance.

### **The size of an UltraLite table is large**

The average time to find a row increases with the size of the table. Therefore, to increase searchability in a very large table, consider using an index. An index allows UltraLite to find rows quickly, but only for columns that are indexed. Otherwise, UltraLite must search every row in the table to see if the row matches the search condition, which can be time consuming in a large table.

### **The UltraLite client application is not performing a large amount of insert, update, or delete operations**

Because UltraLite maintains indexes along with the data itself, an index in this context will have an adverse effect on the performance of database operations. For this reason, you should restrict the use of indexes to data that will be queried regularly described in the point above. Maintaining the UltraLite default indexes (indexes for primary keys and for unique constraints) may be enough.

### **Use indexes on columns involved in WHERE clauses and/or ORDER BY clause**

These indexes can speed the evaluation of these clauses. In particular, an index helps optimize a multi-column ORDER BY clause, but only when the placement of columns in the index and ORDER BY clauses are exactly the same.

## **1.8.2.2 Index Types**

UltraLite supports different types of indexes: unique keys, unique indexes, and non-unique indexes. What differentiates one from the others is what is allowed in that index.

<b>Index characteristic</b>	<b>Unique keys</b>	<b>Unique indexes</b>	<b>Non-unique indexes</b>
Allows duplicate index entries for rows that have the same values in indexed columns.	no	no	yes
Allows null values in index columns.	no	yes	yes

### **i Note**

You can create foreign keys to unique keys, but not to unique indexes.

Also, manually creating an index on a key column is not necessary and generally not recommended. UltraLite creates and maintains indexes for unique keys automatically.

## **Related Information**

[Adding an UltraLite Index \[page 65\]](#)



## 1.8.2.3 Adding an UltraLite Index

Adding indexes to databases speeds up the search process.

### Prerequisites

The database must be connected.

### Context

#### i Note

UltraLite does not detect duplicate or redundant indexes. As indexes must be maintained with the data in your database, add your indexes carefully.

### Procedure

1. Using SQL Central, connect to the UltraLite database.
2. Right-click *Indexes*, and click **► New ► Index ◀**.
3. Follow the instructions in the wizard.

### Results

The index is created.

### Example

To speed up a search on employee surnames in a database that tracks employee information, and tune the performance of queries against this index, you could create an index called *EmployeeNames* and increase the hash size to 20 bytes with the following statement:

```
CREATE INDEX EmployeeNames
ON Employees (Surname, GivenName)
WITH MAX HASH SIZE 20
```

This statement creates an index with the default maximum hash size you have configured. To create an index that overrides the default, ensure you use the WITH MAX HASH SIZE *value* clause to set a new value for this index instance.

## Related Information

[CREATE INDEX Statement \[UltraLite\] \[page 531\]](#)

### 1.8.2.4 Dropping an UltraLite Index

Drop an index from the database.

#### Procedure

1. Using SQL Central, connect to the UltraLite database.
2. In the left pane, double-click *Indexes*.
3. Right-click an index and then click *Delete*.
4. Click *Yes*.

#### Results

The index is removed from the database.

## Related Information

[DROP INDEX Statement \[UltraLite\] \[page 545\]](#)

### 1.8.3 UltraLite Users

A typical UltraLite database contains one user ID and password. UltraLite databases are created with a default user ID of *DBA* and default password of *sql* unless otherwise specified.

Changing the user schema is optional and **not** required. Many applications do not need database-level authentication and assume that a device level password is sufficient authentication to access an application and its data.

Common reasons for not authenticating users may be because the deployment is to a single-user device, or that it is too awkward to prompt a user each time they start the application.

You do not need to include a user ID or password in the database connection string if you do not need database-level authentication. The simplest UltraLite connection string is `DBF= filename`.

When developing an UltraLite application with a custom user authentication interface, you can effectively use the UltraLite user IDs and password hashes stored in an UltraLite database to validate user-supplied credentials and avoid creating your own password hashing algorithm. By adding users to your UltraLite database, you store their user IDs and password hashes. You can then validate the user-supplied credentials in your application by attempting to connect to the database with the UID and PWD connection parameters, where `UID= username` and `PWD= password`. A successful UltraLite database connection indicates that the user is authentic.

### ⚠ Caution

Unlike SQL Anywhere users, UltraLite users are created and managed solely for authentication and not for object ownership or specific database roles and privileges. Once users are authenticated, they gain full access to the database.

By creating user IDs and passwords, you control connections to the UltraLite database but do not secure the data in the database file. The contents are stored as plain text and can be read directly.

To secure the database contents, encrypt the file so that you can authenticate users with an encryption key rather than a user ID and password.

You can obfuscate the file to alter the storage so that data is not stored as plain text, but this approach does not secure the data.

### i Note

UltraLite user IDs are different from MobiLink user names.

## Limitations

The following limitations apply to UltraLite user IDs:

- UltraLite supports up to four unique user IDs per UltraLite database.
- User IDs and passwords can be changed using SQL Central, SQL statements, or UltraLite API methods in your application.
- User IDs have a 31-character limit.
- User IDs cannot include leading single quotes ('), leading double quotes ("), or semicolons(;).
- User IDs are always case insensitive and passwords are always case sensitive.
- User IDs cannot be renamed. You can only add new user IDs and delete existing ones from an existing database connection.
- Users cannot be listed programmatically using the UltraLite APIs. You can only use database tools to list existing users in the database.
- When connecting to an UltraLite database for the first time, the `UID` and `PWD` are the same values that were set when the database was created. UltraLite attempts to connect with the `DBA` user ID and `sql` password when these connection parameters are not specified. You do not need to supply a username or

password when connecting to the database if you did not explicitly set a username and password during its creation.

#### In this section:

##### [Connection Parameters for Managing UltraLite Users \[page 68\]](#)

You can use the *UID* and *PWD* connection parameters to create or authenticate users in an UltraLite database.

##### [SQL Statements for Managing UltraLite Users \[page 70\]](#)

You can use the CREATE USER, ALTER USER, and DROP USER statements to manage users in an UltraLite database.

##### [Creating an UltraLite User with SQL Central \[page 71\]](#)

Use SQL Central to create users for an UltraLite database.

##### [Deleting an UltraLite User with SQL Central \[page 72\]](#)

Use SQL Central to explicitly delete users from an UltraLite database.

## Related Information

[Database Security \[page 35\]](#)

[UltraLite DBKEY Connection Parameter \[page 192\]](#)

[UltraLite UID Connection Parameter \[page 202\]](#)

[UltraLite PWD Connection Parameter \[page 198\]](#)

### 1.8.3.1 Connection Parameters for Managing UltraLite Users

You can use the *UID* and *PWD* connection parameters to create or authenticate users in an UltraLite database.

#### **i** Note

As an alternative to connection parameters, you can use the following UltraLite API methods in your application to grant or revoke user access to an UltraLite database:

- `ULConnection.GrantConnectTo` method [UltraLite C++]
- `ULConnection.RevokeConnectFrom` method [UltraLite C++]
- `ULGrantConnectTo` method [UltraLite Embedded SQL]
- `ULRevokeConnectFrom` method [UltraLite Embedded SQL]
- `ULConnection.GrantConnectTo` method [UltraLite.NET]
- `ULConnection.RevokeConnectFrom` method [UltraLite.NET]

Grant and revoke methods are not available in the UltraLiteJ API. For Android devices, set the UID and PWD connection parameters using the `ConfigPersistent.setConnectionString` method.

## Use Connection Parameters with UltraLite Databases

For most UltraLite APIs, the `createDatabase` method of a `DatabaseManager` object can be used to create a new UltraLite database with the specified connection and creation parameters.

The following example illustrates how to create a default user for a new UltraLite database by passing the *UID* and *PWD* parameters to the `CreateDatabase` method in the UltraLite C++ API:

```
ULConnection * conn;
ULError ulerr;
ULDatabaseManager::CreateDatabase("dbf=sample.udb;uid=default-name;pwd=default-
password", &ulerr);
```

The following example illustrates how to authenticate a user in an existing UltraLite database by passing the *UID* and *PWD* parameters to the `OpenConnection` method in the UltraLite C++ API:

```
ULConnection * conn;
ULError ulerr;
ULDatabaseManager::OpenConnection("dbf=sample.udb;uid=test-name;pwd=test-
password", &ulerr);
```

## Use Connection Parameters with UltraLite Databases on Android Devices

For the UltraLiteJ API, you use the `setConnectionString` method of a `Configuration` object in the UltraLiteJ API to create or authenticate users.

The following example illustrates how to create a default user for a new UltraLite database by passing the *UID* and *PWD* parameters to the `createDatabase` method in the UltraLiteJ API:

```
ConfigFile config =
    DatabaseManager.createConfigurationFileAndroid("DBname.udb",
getApplicationContext());
config.setConnectionString("uid=default-name;pwd=default-password");
Connection conn = DatabaseManager.createDatabase(config);
```

The following example illustrates how to authenticate a user in an existing UltraLite database by passing the *UID* and *PWD* parameters and the `connect` method in the UltraLiteJ API:

```
ConfigFile config =
    DatabaseManager.createConfigurationFileAndroid("DBname.udb",
getApplicationContext());
config.setConnectionString("uid=test-name;pwd=test-password");
Connection conn = DatabaseManager.connect(config);
```

As an alternative to the `setConnectionString` method, you can use the `setPassword` or `setUserName` methods to create or authenticate a user, respectively.

## Related Information

[UltraLite UID Connection Parameter \[page 202\]](#)

## 1.8.3.2 SQL Statements for Managing UltraLite Users

You can use the CREATE USER, ALTER USER, and DROP USER statements to manage users in an UltraLite database.

### i Note

As an alternative to SQL statements, you can use the following UltraLite API methods in your application to grant or revoke user access to an UltraLite database:

- `ULConnection.GrantConnectTo` method [UltraLite C++]
- `ULConnection.RevokeConnectFrom` method [UltraLite C++]
- `ULGrantConnectTo` method [UltraLite Embedded SQL]
- `ULRevokeConnectFrom` method [UltraLite Embedded SQL]
- `ULConnection.GrantConnectTo` method [UltraLite.NET]
- `ULConnection.RevokeConnectFrom` method [UltraLite.NET]

Grant and revoke methods are not available in the UltraLiteJ API. For Android devices, construct a CREATE USER, ALTER USER, or DROP USER statement as a string variable and pass it to the `Connection.prepareStatement` method.

### Example

The following example illustrates how to use the UltraLiteJ API connect to an existing UltraLite database, and use the CREATE USER statement to create a new user:

```
ConfigFile config =
    DatabaseManager.createConfigurationFileAndroid("DBname.udb",
getApplicationContext());
Connection conn = DatabaseManager.connect(config);
String sql_string = "CREATE USER test-user IDENTIFIED BY test-password";
PreparedStatement authenticator = conn.prepareStatement(sql_string);
authenticator.execute();
authenticator.close();
```

### Related Information

[CREATE USER Statement \[UltraLite\] \[page 543\]](#)

[ALTER USER Statement \[UltraLite\] \[page 528\]](#)

[DROP USER Statement \[UltraLite\] \[page 549\]](#)

## 1.8.3.3 Creating an UltraLite User with SQL Central

Use SQL Central to create users for an UltraLite database.

### Context

#### i Note

As an alternative to SQL Central, you can use the following UltraLite API methods in your application to grant user access to an UltraLite database:

- `ULConnection.GrantConnectTo` method [UltraLite C++]
- `ULGrantConnectTo` method [UltraLite Embedded SQL]
- `ULConnection.GrantConnectTo` method [UltraLite.NET]

Grant methods are not available in the UltraLiteJ API. For Android devices, construct a `CREATE USER` statement as a string variable and pass it to the `Connection.prepareStatement` method.

### Procedure

1. Using SQL Central, connect to the UltraLite database.
2. Right-click the *Users* folder, and click ► *New* ► *User* ▾.
3. Follow the instructions in the wizard.

### Results

The new user is created.

### Related Information

[CREATE USER Statement \[UltraLite\] \[page 543\]](#)

## 1.8.3.4 Deleting an UltraLite User with SQL Central

Use SQL Central to explicitly delete users from an UltraLite database.

### Context

#### i Note

As an alternative to SQL Central, you can use the following UltraLite API methods in your application to revoke user access to an UltraLite database:

- `ULConnection.RevokeConnectFrom` method [UltraLite C++]
- `ULRevokeConnectFrom` method [UltraLite Embedded SQL]
- `ULConnection.RevokeConnectFrom` method [UltraLite.NET]

Revoke methods are not available in the UltraLiteJ API. Construct a `DROP USER` statement as a string variable and pass it to the `Connection.prepareStatement` method.

### Procedure

1. Using SQL Central, connect to the UltraLite database.
2. In the left pane, double-click the *Users* folder.
3. Right-click the user and click *Delete*.

### Results

The user is deleted from the database.

### Related Information

[DROP USER Statement \[UltraLite\] \[page 549\]](#)

## 1.9 UltraLite as a MobiLink Client

You can configure UltraLite to act as a MobiLink client.



**In this section:**

[UltraLite Clients \[page 73\]](#)

Synchronizing an UltraLite database requires your application to set synchronization parameters.

[Microsoft ActiveSync Synchronization Overview \[page 92\]](#)

Synchronization through Microsoft ActiveSync can be summarized in a few steps.

[UltraLite Synchronization Parameters \[page 93\]](#)

Synchronization parameters control the synchronization between an UltraLite database and the MobiLink server.

[UltraLite Network Protocol Options \[page 122\]](#)

You must set the network protocol in your application.

## 1.9.1 UltraLite Clients

Synchronizing an UltraLite database requires your application to set synchronization parameters.

These parameters identify the address of the MobiLink server and other required information, and calling a synchronization function or executing the SYNCHRONIZE SQL statement. The option you chose depends on the API you are using.

**In this section:**

[UltraLite Client Synchronization Behavior Customization \[page 73\]](#)

Adding custom synchronization support to UltraLite involves several tasks.

[Primary Key Uniqueness in UltraLite \[page 75\]](#)

UltraLite can maintain primary key uniqueness using any of the techniques supported by MobiLink.

[UltraLite Client Synchronization Design \[page 79\]](#)

All data in an UltraLite database is synchronized by default.

[MobiLink File Transfers \[page 88\]](#)

UltraLite supports the ability to transfer files with the MobiLink server.

[UltraLite Publications \[page 89\]](#)

A publication is a database object that identifies the data that is to be synchronized.

### 1.9.1.1 UltraLite Client Synchronization Behavior Customization

Adding custom synchronization support to UltraLite involves several tasks.

**Maintain primary key uniqueness in synchronization models that include more than one remote client**

Required. In a synchronization system, the primary key is the only way to identify the same row in different databases (remote and consolidated) and the only way to detect conflicts. Therefore, multiple clients must adhere to the following rules:

- Every table that is to be synchronized must have a primary key.
- Never update the values of primary keys.
- Primary keys must be unique across all synchronized databases.

#### **Ensure your date columns are set up so that fractional data is not lost**

For a SQL Anywhere consolidated database this is not typically an issue. However, for databases like Oracle, there may be compatibility issues that you need to consider. For example, UltraLite and Oracle databases must share the same timestamp precision. Additionally, you should also add a `TIMESTAMP` to the Oracle database to avoid losing fractional second data when the UltraLite remote databases uploads data to the consolidated database.

#### **Describe what data subsets you want to upload to the consolidated database**

Optional. You only need to do this when you do not want to synchronize all data by default. To target what data you want to synchronize, use one or more subsetting techniques.

For example, you may want to create a publication for high-priority data. The application could then synchronize this data over wireless networks. Because wireless networks can have high usage costs associated with them, you may want to limit these usage fees to those that are business critical. You can then synchronize less time-sensitive data from a cradle at a later time.

#### **Initialize synchronization from your UltraLite application and supply the parameters that describe the session**

Required. Programming synchronization has two parts: describing the session, and then initiating the synchronization operation.

Describing the session primarily involves choosing a synchronization communication stream (also known as a network protocol), and the parameters for that stream, setting the version of your synchronization scripts, and identifying the MobiLink user. However, there are other parameters you can set: for example, use the `upload_only` and `download_only` parameters to change the default bi-directional synchronization to one-way only.

All other important synchronization behaviors are controlled at the MobiLink server with MobiLink synchronization scripts. These include:

- What data is downloaded as updates or inserts to tables in the UltraLite remote.
- What processing is required on uploaded changes from a remote database.

You can write your synchronization scripts so that data is partitioned among remote databases in an appropriate manner.

## **Related Information**

[Unique Primary Keys](#)

[Primary Key Uniqueness in UltraLite \[page 75\]](#)

[Oracle Consolidated Database](#)

[UltraLite Client Synchronization Design \[page 79\]](#)

[Synchronization Setup for Your UltraLite Application \[page 86\]](#)

[MobiLink Consolidated Databases](#)  
[Synchronization Scripts](#)  
[Direct Row Handling](#)  
[Partitioned Rows Among Remote Databases](#)  
[UltraLite Precision Creation Option \[page 168\]](#)

## 1.9.1.2 Primary Key Uniqueness in UltraLite

UltraLite can maintain primary key uniqueness using any of the techniques supported by MobiLink.

One of these methods is to use a GLOBAL AUTOINCREMENT column. GLOBAL AUTOINCREMENT is similar to AUTOINCREMENT, except that the domain is partitioned. UltraLite supplies column values only from the partition assigned to the database's global database ID. Each UltraLite database is assigned a unique integer global database ID.

A second method is to use a UUID primary key column. A UUID requires more data, but needs no distinct database identifier.

### In this section:

#### [GLOBAL AUTOINCREMENT Columns in UltraLite \[page 75\]](#)

You can declare the default value of a column in an UltraLite database to be of type GLOBAL AUTOINCREMENT.

#### [Methods for Finding the Last Assigned GLOBAL AUTOINCREMENT Value \[page 77\]](#)

You can retrieve the GLOBAL AUTOINCREMENT value that was chosen during the most recent insert operation.

#### [Partition Sizes \[page 77\]](#)

The partition size can be any positive integer but should be set so that the supply of numbers within any one partition is not likely to be exhausted.

## Related Information

[Unique Primary Keys](#)  
[UltraLite global\\_database\\_id Option \[page 210\]](#)

### 1.9.1.2.1 GLOBAL AUTOINCREMENT Columns in UltraLite

You can declare the default value of a column in an UltraLite database to be of type GLOBAL AUTOINCREMENT.

Before you can autoincrement these column IDs, you must first set the global database ID for the UltraLite database.

## ⚠ Caution

GLOBAL AUTOINCREMENT column values downloaded via MobiLink synchronization do not update the GLOBAL AUTOINCREMENT value counter. As a result, an error can occur should one MobiLink client insert a value into another client's partition. To avoid this problem, ensure that each copy of your UltraLite application inserts values only in its own partition.

The `global_database_id` database option allows you to set the value in your UltraLite database. When deploying UltraLite, you must assign a different identification number to each database.

Allow UltraLite to supply default values for the column using the partition uniquely identified by the UltraLite database's number.

UltraLite follows these rules:

- If the column contains no values in the current partition, the first default value is  $p \cdot n + 1$ .  $p$  represents the partition size and  $n$  represents the global ID number.
- If the column contains values in the current partition, but all are less than  $p \cdot (n + 1)$ , the next default value will be one greater than the previous maximum value in this range.
- Default column values are not affected by values in the column outside the current partition; that is, by numbers less than  $p \cdot n + 1$  or greater than  $p \cdot (n + 1)$ . Such values may be present if they have been replicated from another database via MobiLink synchronization.  
For example, if you assigned your UltraLite database a global ID of 1 and the partition size is 1000, then the default values in that database would be chosen in the range 1001-2000. Another copy of the database, assigned the identification number 2, would supply default values for the same column in the range 2001-3000.
- Because you cannot set the global ID number to a negative value, the GLOBAL AUTOINCREMENT column values are always positive. The maximum identification number is restricted only by the column data type and the partition size.
- If you do not set a global ID value, or if you exhaust values from the partition, a NULL value is inserted into the column. Should NULL values not be permitted, the attempt to insert the row causes an error.

If you exhaust or will soon exhaust available values for columns declared as GLOBAL AUTOINCREMENT, you need to set a new global database ID. UltraLite chooses GLOBAL AUTOINCREMENT values from the partition identified by the global ID number, but only until the maximum value is reached. If you exceed values, UltraLite begins to generate NULL values. By assigning a new global database ID number, you allow UltraLite to set appropriate values from another partition.

One approach of choosing a new global database ID is to maintain a pool of unused global database ID values. This pool is maintained in the same manner as a pool of primary keys.

## i Note

UltraLite APIs provide means of obtaining the proportion of numbers that have been used. The return value is a SHORT in the range 0-100 that represents the percent of values used so far. For example, a value of 99 indicates that very few unused values remain and the database should be assigned a new identification number. The method of setting this identification number varies according to the programming interface you are using.

## Related Information

[Primary Key Pools with SQL Remote](#)

[Partition Sizes \[page 77\]](#)

[UltraLite global\\_database\\_id Option \[page 210\]](#)

[UltraLite global\\_database\\_id Option \[page 210\]](#)

### 1.9.1.2.2 Methods for Finding the Last Assigned GLOBAL AUTOINCREMENT Value

You can retrieve the GLOBAL AUTOINCREMENT value that was chosen during the most recent insert operation.

Since these values are often used for primary keys, knowing the generated value may let you more easily insert rows that reference the primary key of the first row. You can check the value with:

#### UltraLite for C/C++

Use the GetLastIdentity function on the ULConnection object.

#### UltraLite.NET

Use the LastIdentity property on the ULConnection object.

#### UltraLiteJ

Use the getLastIdentity method on the Connection object.

The returned value is an unsigned 64-bit integer, database data type UNSIGNED BIGINT. Since this statement 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.

#### i Note

Occasionally, a single INSERT statement may include more than one column of type GLOBAL AUTOINCREMENT. In this case, the return value 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 in a way that avoids this situation.

### 1.9.1.2.3 Partition Sizes

The partition size can be any positive integer but should be set so that the supply of numbers within any one partition is not likely to be exhausted.

For columns of type INT or UNSIGNED INT, the default partition size is  $2^{16} = 65536$ ; for columns of other types the default partition size is  $2^{32} = 4294967296$ . Since these defaults may be inappropriate, it is best to specify the partition size explicitly.

Default partition sizes for some data types are different in UltraLite applications than in SQL Anywhere databases. Declare the partition size explicitly if you want different databases to remain consistent.

## In this section:

[Overriding the Partition Size for a GLOBAL AUTOINCREMENT Column \[page 78\]](#)

Override the partition size of a GLOBAL AUTOINCREMENT column.

## Related Information

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

[ALTER TABLE Statement \[UltraLite\] \[page 524\]](#)

### 1.9.1.2.3.1 Overriding the Partition Size for a GLOBAL AUTOINCREMENT Column

Override the partition size of a GLOBAL AUTOINCREMENT column.

## Prerequisites

You must be connected to an UltraLite database.

## Context

Increasing the partition size of a GLOBAL AUTOINCREMENT column ensures that the supply of numbers within the partition is rarely exhausted.

## Procedure

1. Create a table with a GLOBAL AUTOINCREMENT column that has a partition size specified in parentheses.

Execute the following SQL code:

```
CREATE TABLE customer (  
  id INT DEFAULT GLOBAL AUTOINCREMENT (5000),  
  name VARCHAR(128) NOT NULL,  
  PRIMARY KEY (id)  
)
```

A simple reference table with two columns: an integer that holds a customer identification number, and a character string that holds the customer's name is created. The ID has a partition size of 5000.

2. Connect to the UltraLite database in SQL Central.
3. Right-click the ID column of the customer table and click *Properties*.
4. Click the *Value* tab.
5. Enter a positive integer that is greater than 5000 in the *Partition Size* field.

## Results

The partition size of the ID column is updated according to the value entered in the *Partition Size* field.

### 1.9.1.3 UltraLite Client Synchronization Design

All data in an UltraLite database is synchronized by default.

If you are new to deploying UltraLite as a MobiLink remote database, plan to use the default behavior initially.

Once you become comfortable with the synchronization process, you may decide to customize the behavior of the synchronization operation to capture more complex business logic. Designing custom synchronization behavior requires that you ask the following questions. If your business requirements are simple, you may only need to use a single synchronization feature. However, in very complex deployments, you may need to use multiple synchronization features to configure the desired synchronization behavior.

Design question	If you answer yes, use the following
Do you want to download changes from the consolidated database but not have local changes uploaded to the database?	The download_only table name suffix allows you to identify any tables for which the synchronization should be download only. Changes made to the local tables are not uploaded to the consolidated database.
Do you want to exclude tables from synchronization?	The nosync table name suffix allows you to identify any tables that you do not want to synchronize.
Do you only want to synchronize entire tables even when data hasn't changed?	The allsync table name suffix allows you to synchronize the entire table, even when no changes are detected.
Do you want to synchronize an entire table or just rows that meet specific conditions? Does some of the data require synchronization priority due to its importance or time-sensitivity?	<p>A publication includes articles that list the tables that require synchronization. An article can include a WHERE clause that specifies the rows to upload based on whether the rows meet the defined criteria.</p> <p>Multiple publications can address priority issues that require certain UltraLite data be uploaded before others.</p>
Do you want a table order for synchronization because you have cycles of foreign keys?	The Table Order synchronization parameter allows you to determine the order of synchronization operations when you have foreign key cycles. However, foreign key cycles are generally not recommended for UltraLite.

**Design question****If you answer yes, use the following**

---

Do you want to control synchronization behavior? For example, do you need downloads to occur at the same time as uploads? Or do you want to change bi-directional synchronization to one-way only?

Use the appropriate synchronization parameter as part of:

- Your application's synchronization structure (or the synchronization enumeration).
- The `ulsync` utility's `-e` option.

---

Do you want your UltraLite client to be TLS-enabled?

What encryption algorithm you choose determines how your device must be set up according to the platform that runs on that device.

---

**In this section:**[UltraLite Non-synchronizing Tables \[page 81\]](#)

By creating the table using `SYNCHRONIZE OFF`, you control when to exclude the entire table from the upload operation.

[UltraLite Download-only Tables \[page 82\]](#)

By creating the table using `SYNCHRONIZE DOWNLOAD`, you exclude entire tables from the upload operation.

[UltraLite Synchronize-All Tables \[page 82\]](#)

By creating or altering a table using `SYNCHRONIZE ALL`, you control whether to change the synchronization behavior during upload so that it synchronizes all table data, even if nothing has changed since the previous synchronization session.

[Publishing Data in UltraLite \[page 83\]](#)

Add publications to an UltraLite database using SQL Central or SQL.

[Table Order in UltraLite \[page 84\]](#)

By setting the Table Order synchronization parameter you can control the order of synchronization operations.

[Synchronization Setup for Your UltraLite Application \[page 86\]](#)

In UltraLite, synchronization begins by opening a specific connection with the MobiLink server over the configured communication stream (also known as a network protocol).

**Related Information**

[UltraLite Synchronization Parameters \[page 93\]](#)

[UltraLite Network Protocol Options \[page 122\]](#)

[The Synchronization Process](#)

[The Synchronization Process](#)



## 1.9.1.3.1 UltraLite Non-synchronizing Tables

By creating the table using SYNCHRONIZE OFF, you control when to exclude the entire table from the upload operation.

You can use these non-synchronizing tables for client-specific persistent data that is not required in the consolidated database. Other than being excluded from synchronization, you can use these tables in exactly the same way as other tables in the UltraLite database.

### i Note

The synchronization type for a table can only be changed if it does not have any unsynchronized changes that need to be uploaded.

If you create a table with a `_nosync` suffix, you can only rename that table so it retains the `_nosync` suffix. For example, the following ALTER TABLE statement with a rename clause is not allowed because the new name no longer ends in `nosync`:

```
ALTER TABLE purchase_comments_nosync
RENAME comments
```

To correct this, the statement must be rewritten to include this suffix:

```
ALTER TABLE purchase_comments_nosync
RENAME comments_nosync
```

You can alternatively use publications to achieve the same effect.

### i Note

As an alternative to creating or altering a table with the SYNCHRONIZE OFF clause, you can append the phrase `_nosync` to the table name to turn it into a non-synchronizing table.

## Related Information

### Publications

[UltraLite Publications \[page 89\]](#)

### Synchronization Scripts

[Download Only Synchronization Parameter \[page 101\]](#)

[Upload Only Synchronization Parameter \[page 118\]](#)

[Additional Parameters Synchronization Parameter \[page 95\]](#)

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

[ALTER TABLE Statement \[UltraLite\] \[page 524\]](#)

[CREATE PUBLICATION Statement \[UltraLite\] \[page 533\]](#)

## 1.9.1.3.2 UltraLite Download-only Tables

By creating the table using SYNCHRONIZE DOWNLOAD, you exclude entire tables from the upload operation.

You can use these tables for data that should not be synchronized to the consolidated database. Other than being excluded from synchronization, you can use these tables in exactly the same way as other tables in the UltraLite database.

You can alternatively use publications to achieve the same effect.

### i Note

The synchronization type for a table can only be changed if it does not have any unsynchronized changes that need to be uploaded.

### i Note

As an alternative to creating or altering a table with the SYNCHRONIZE DOWNLOAD clause, you can append the phrase *\_download\_only* to the table name to turn it into a download-only table.

## Related Information

[Publications](#)

[UltraLite Publications \[page 89\]](#)

[Synchronization Scripts](#)

[Download Only Synchronization Parameter \[page 101\]](#)

[Upload Only Synchronization Parameter \[page 118\]](#)

[Additional Parameters Synchronization Parameter \[page 95\]](#)

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

[ALTER TABLE Statement \[UltraLite\] \[page 524\]](#)

[CREATE PUBLICATION Statement \[UltraLite\] \[page 533\]](#)

## 1.9.1.3.3 UltraLite Synchronize-All Tables

By creating or altering a table using SYNCHRONIZE ALL, you control whether to change the synchronization behavior during upload so that it synchronizes all table data, even if nothing has changed since the previous synchronization session.

### i Note

The synchronization type for a table can only be changed if it does not have any unsynchronized changes that need to be uploaded.

Some UltraLite applications require user/client-specific data that you can store in a SYNCHRONIZE ALL TABLES. You can upload the data in the table to a temporary table in the consolidated database, use the data to

control synchronization by your other scripts without having the data maintained in the consolidated database. For example, you may want your UltraLite applications to indicate which channels or topics they are interested in, and use this information to download the appropriate rows.

### **i** Note

As an alternative to creating or altering a table with the SYNCHRONIZE ALL clause, you can append the phrase `_allsync` to the table name to turn it into a synchronize-all table.

## **Related Information**

[Publications](#)

[UltraLite Publications \[page 89\]](#)

[Synchronization Scripts](#)

[Download Only Synchronization Parameter \[page 101\]](#)

[Upload Only Synchronization Parameter \[page 118\]](#)

[Additional Parameters Synchronization Parameter \[page 95\]](#)

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

[ALTER TABLE Statement \[UltraLite\] \[page 524\]](#)

[CREATE PUBLICATION Statement \[UltraLite\] \[page 533\]](#)

### **1.9.1.3.4 Publishing Data in UltraLite**

Add publications to an UltraLite database using SQL Central or SQL.

## **Context**

Publications define a set of articles that describe the data to be synchronized.

Publication articles can be a whole table, or can define a subset of the data in a table. You can include an optional predicate (a WHERE clause) to define a subset of rows from a given table. Publications are more flexible than creating tables with SYNCHRONIZE OFF. To synchronize data subsets of an UltraLite database *separately*, use multiple publications. You can then combine publications with upload-only or download-only synchronization parameters to synchronize high-priority changes efficiently.

### **i** Note

The maximum number of user publications in UltraLite is 63.

UltraLite publications do not support the definition of column subsets, nor the SUBSCRIBE BY clause that is available in SQL Anywhere. If columns in an UltraLite table do not exactly match tables in a consolidated database, use MobiLink scripts to resolve those differences.

You do not need to set a table synchronization order in a publication. If table order is important for your deployment, you can set the table order when you synchronize the UltraLite database by setting the Table Order synchronization parameter.

## Procedure

1. Connect to the UltraLite database using the UltraLite plug-in.
2. Right-click the *Publications* folder and click **► New ► Publication ►**.
3. Enter a name for the new publication. Click *Next*.
4. On the *Tables* tab, click a table in the *Matching Tables* list. Click *Add*.

The table appears in the *Selected Tables* list on the right.

5. Add additional tables.
6. If necessary, click the *Where* tab to specify the rows to be included in the publication. You cannot specify column subsets.
7. Click *Finish*.

## Results

The new publication is created.

## Related Information

[UltraLite Publications \[page 89\]](#)

[Publications](#)

[Synchronization Scripts](#)

[Download Only Synchronization Parameter \[page 101\]](#)

[Upload Only Synchronization Parameter \[page 118\]](#)

[Additional Parameters Synchronization Parameter \[page 95\]](#)

[CREATE PUBLICATION Statement \[UltraLite\] \[page 533\]](#)

### 1.9.1.3.5 Table Order in UltraLite

By setting the Table Order synchronization parameter you can control the order of synchronization operations.

To specify a table order for synchronization, you can use the TableOrder parameter programmatically or as part of the ulsync utility during testing. The TableOrder parameter specifies the order of tables that are to be uploaded.

You only need to explicitly set the table order if your UltraLite database has:

- Foreign key cycles. You must then list all tables that are part of a cycle.
- Different foreign key relationships from those used in the consolidated database.

**In this section:**

[Avoiding Synchronization Issues with Foreign Key Cycles \[page 85\]](#)

Table order is particularly important for UltraLite databases that use foreign key cycles.

## Related Information

[Referential Integrity and Synchronization](#)

[Additional Parameters Synchronization Parameter \[page 95\]](#)

### 1.9.1.3.5.1 Avoiding Synchronization Issues with Foreign Key Cycles

Table order is particularly important for UltraLite databases that use foreign key cycles.

A cycle occurs when you link a series of tables together such that a circle is formed. However, due to complexities that arise when cycles between the consolidated database and the UltraLite remote database differ, foreign key cycles are not recommended.

With foreign key cycles, you should order your tables so that operations for a primary table come before the associated foreign table. A Table Order parameter ensures that the insert in the foreign table will have its foreign key referential integrity constraint satisfied (likewise for other operations like delete).

In addition to table ordering, another method you can use to avoid synchronization issues is to postpone the checking of referential integrity until the transaction is committed. If your consolidated database is a SQL Anywhere database, use the *CHECK ON COMMIT* clause on one of the foreign keys. This ensures that foreign key referential integrity is checked during the commit phase rather than when the operation is initiated. For example:

```
CREATE TABLE c (  
    id INTEGER NOT NULL PRIMARY KEY,  
    c_pk INTEGER NOT NULL  
);  
CREATE TABLE p (  
    pk INTEGER NOT NULL PRIMARY KEY,  
    c_id INTEGER NOT NULL,  
    FOREIGN KEY p_to_c (c_id) REFERENCES c(id)  
);  
ALTER TABLE c  
    ADD FOREIGN KEY c_to_p (c_pk)  
    REFERENCES p(pk)  
    CHECK ON COMMIT;
```

If your consolidated database is from another database vendor, check to see if the database has similar methods of checking referential integrity. If so, you should implement this method. Otherwise, you must redesign table relationships to eliminate all foreign key cycles.

## Related Information

[Referential Integrity and Synchronization](#)

[Additional Parameters Synchronization Parameter \[page 95\]](#)

### 1.9.1.3.6 Synchronization Setup for Your UltraLite Application

In UltraLite, synchronization begins by opening a specific connection with the MobiLink server over the configured communication stream (also known as a network protocol).

In addition to synchronization support for direct network connections, Microsoft Windows Mobile devices also support Microsoft ActiveSync synchronization.

## Defining the Connection

Each UltraLite remote database that synchronizes with a MobiLink server does so over a network protocol. You set the network protocol with the synchronization stream parameter. Supported network protocols include TCP/IP, HTTP, HTTPS, and TLS. For the protocol you choose, you also need to supply stream parameters that define other required connection information like the MobiLink server host and the port. You must also supply the MobiLink user information and the synchronization script version.

## Defining the Synchronization Behavior

You can control synchronization behavior by setting various synchronization parameters. The way you set parameters depends on the specific UltraLite interface you are using.

Important behaviors to consider include:

### **Synchronization direction**

By default, synchronization is bi-directional. When using only one-way synchronizations, remember to use the appropriate `upload_only` or `download_only` parameter. By performing one-way synchronizations, you minimize the synchronization time required. Also, with download-only synchronization, you do not have to commit all changes to the UltraLite database before synchronization. Uncommitted changes to tables not involved in synchronization are not uploaded, so incomplete transactions do not cause problems.

To use download-only synchronization, you must ensure that rows overlapping with the download are not changed locally. If any data is changed locally, synchronization fails in the UltraLite application with a `SQL_E_DOWNLOAD_CONFLICT` error.

### **Concurrent changes during synchronization**

During the upload phase, UltraLite applications can access UltraLite databases in a read-only fashion. During the download phase, read-write access is permitted, but if an application changes a row that the

download then attempts to change, the download will fail and roll back. You can disable concurrent access to data during synchronization by setting the `DisableConcurrency` synchronization parameter.

The following procedure is generally used to add synchronization functionality to your application:

1. Supply the necessary synchronization parameters and protocol options needed for the session as fields of a synchronization information structure.

For example, using the UltraLite C++ API, you add synchronization to your application by setting appropriate values in the `ul_sync_info` structure:

```
ul_sync_info info;
// define a sync structure named "info"
ULEnableTcpipSynchronization( &sqlca );
// use a TCP/IP stream
conn->InitSynchInfo( &info );
// initialize the structure
info.stream = ULStream();
// specify the Socket Stream
info.stream_parms= UL_TEXT( "host=myMLserver;port=2439" );
// set the MobiLink host information
info.version = UL_TEXT( "custdb 11.0" );
// set the MobiLink version information
info.user_name = UL_TEXT( "50" );
// set the MobiLink user name
info.download_only =ul_true;
// make the synchronization download-only
```

2. Initialize synchronization.

For direct synchronization, you would call an API-specific synchronization method. These methods return a boolean value, indicating success or failure of the synchronization operation. If the synchronization fails, you can examine detailed error status fields in another structure to get additional error information.

For Microsoft ActiveSync synchronization, you must catch the synchronization message from the Microsoft ActiveSync provider and use the `DoSync` method to call the `ULSynchronize` method.

3. Use an observer callback function to report the progress of the synchronization to the user.

### Note

If you have an environment where DLLs fail either because the download is very large or the network connection is unreliable, you may want to implement resumable downloads.

## Related Information

[Failed Downloads](#)

[Resumption of Failed Downloads](#)

[Upload-only and Download-only Synchronizations](#)

[The Synchronization Process](#)

[UltraLite Synchronization Parameters \[page 93\]](#)

[UltraLite Network Protocol Options \[page 122\]](#)

[MobiLink Data Synchronization in UltraLite.NET \[page 620\]](#)

[MobiLink Data Synchronization in UltraLite C++ \[page 666\]](#)

[Microsoft ActiveSync Synchronization in UltraLite C++ \[page 713\]](#)

[Synchronization Setup for an Embedded SQL Application \[page 699\]](#)

## 1.9.1.4 MobiLink File Transfers

UltraLite supports the ability to transfer files with the MobiLink server.

For all other APIs, use the MobiLink file transfer mechanism when:

- You have multiple files that you need to deploy to multiple devices, particularly when corporate firewalls are used as a security measure. Because MobiLink is already configured to handle synchronization through these firewalls, the MLFileTransfer mechanism makes device provisioning for upgrades and other types of file transfers very convenient.
- You have files that you want to target to a specific MobiLink user ID. This requires that you create one or more user-specific directories on the MobiLink server for each user ID. Otherwise, if you only have a single version of the file, you can use a default directory.

### How File Transfers Work

You can employ one of two MobiLink-initiated file transfer mechanisms to download files to a device: run the `mfiletransfer` utility for desktop transfers, or call the appropriate function for the API you are using to code your UltraLite application. Both approaches require that you:

1. Describe the transfer destination.  
Whether you use the `mfiletransfer` utility from the desktop, or whether you use the function appropriate to your API, you must set the local path and file name of the file on the target device or desktop computer. If none are supplied in the application or by the end user, then the source file name is assumed and the file is stored in the current directory.  
The destination directory of the target can vary depending on the device's operating system:
  - On Microsoft Windows Mobile, if the destination is NULL, the file is stored in the root directory ( \ ). The file name must follow file name conventions for Microsoft Windows Mobile.
  - On the desktop, if the destination is NULL, the file is stored in the current directory. The file name must follow file name conventions for the desktop system.
  - On Apple iOS, you should store files in your application's document directory. You can get the location of the document directory by calling the `NSSearchPathForDirectories/uDomains` using the `NSDocumentDirectory` parameter.
2. Set the MobiLink user credentials that allow the user to be identified and the correct file(s) to be downloaded.  
This user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink server.
3. Set the stream type you want to use, and define the parameters for the desired stream. These are the same parameters supported by UltraLite for MobiLink synchronization.  
Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. If you set the stream type to a value that is invalid for the platform, the stream type is set to TCP/IP.
4. Describe the required behavior for the transfer mechanism.  
For example, you can set properties that allow this mechanism to force a download even when the file already exists on the target and has not changed, or that allow partial downloads to be resumed. You can also set whether you want the progress to be monitored and reported upon.
5. Ensure the MobiLink server is running and has been started with the `-ftr` option.



6. Start the transfer, and, if applicable, monitor the download progress.  
By displaying the download progress, the user can cancel and resume the download at a later time.

## Related Information

[UltraLite File Path Formats in Connection Parameters \[page 42\]](#)

[UltraLite Synchronization Parameters \[page 93\]](#)

[UltraLite Network Protocol Options \[page 122\]](#)

[-ftr mlsrv17 Option](#)

[MobiLink File Transfer Utility \(mfiletransfer\)](#)

### 1.9.1.5 UltraLite Publications

A publication is a database object that identifies the data that is to be synchronized.

To synchronize all tables and all rows of those tables in your UltraLite database, do not create any publications.

A publication consists of a set of articles. Each article may be an entire table, or may be rows in a table. You can define this set of rows with a WHERE clause.

Each database can have multiple publications, depending on the desired synchronization logic. For example, you may want to create a publication for high-priority data. The user can synchronize this data over high-speed wireless networks. Because wireless networks can have usage costs associated with them, you would want to limit these usage fees to those that are business-critical only. All other less time-sensitive data could be synchronized from a cradle at a later time.

You create publications using SQL Central or with the CREATE PUBLICATION statement. In SQL Central, all publications and articles appear in the *Publications* folder.

## Usage Notes

- UltraLite publications do not support the definition of column subsets, nor the SUBSCRIBE BY clause. If columns in an UltraLite table do not exactly match tables in a SQL Anywhere consolidated database, use MobiLink scripts to resolve those differences.
- Columns are always sent in the order in which they were defined in the CREATE TABLE statement.
- You do not need to set a table synchronization order in a publication. If table order is important for your deployment, you can set the table order when you synchronize the UltraLite database by setting the Table Order synchronization parameter.
- Because object ownership is not supported in UltraLite, any user can delete a publication.

### In this section:

[Publishing Whole Tables in UltraLite \[page 90\]](#)

Publish an entire table in UltraLite

[Publishing a Subset of Rows from an UltraLite Table \[page 91\]](#)

Publish a subset of rows from an UltraLite table

[Dropping a Publication for UltraLite \[page 92\]](#)

Drop a publication for UltraLite.

## Related Information

[Table Order in UltraLite \[page 84\]](#)

[Publications](#)

[UltraLite Client Synchronization Design \[page 79\]](#)

[Synchronization Scripts](#)

### 1.9.1.5.1 Publishing Whole Tables in UltraLite

Publish an entire table in UltraLite

#### Context

A publication consists of a set of articles. The simplest publication you can make consists of a single article, which consists of all rows and columns of a table.

You can use either SQL Central or Interactive SQL to perform this task.

#### Procedure

1. Using SQL Central, connect to the UltraLite database.
2. Right-click the *Publications* folder, and click **► New ► Publication ▾**.
3. In the *What Do You Want To Name The New Publication* field, type a name for the new publication. Click *Next*.
4. On the *Tables* tab, click tables in the *Available Tables* list. Click *Add*.
5. Click *Finish*.

#### Results

The publication is created.

## Related Information

[UltraLite Clients \[page 73\]](#)

[CREATE PUBLICATION Statement \[UltraLite\] \[page 533\]](#)

### 1.9.1.5.2 Publishing a Subset of Rows from an UltraLite Table

Publish a subset of rows from an UltraLite table

#### Context

A publication can only contain specific table rows. A WHERE clause limits the rows that are uploaded to those that have changed and satisfy a search condition in the WHERE clause.

To upload all changed rows, do not specify a WHERE clause.

#### Procedure

1. Using SQL Central, connect to the UltraLite database.
2. Right-click the *Publications* folder, and click **► New ► Publication ▾**.
3. In the *What Do You Want To Name The New Publication* field, type a name for the new publication.
4. Click *Next*.
5. In the *Available Tables* list, click a table and click *Add*.
6. Click the *WHERE Clauses* tab, and click the table from the *Articles* list. Optionally, you can use the *Insert* window to assist you in formatting the search condition.
7. Click *Finish*.

#### Results

The rows that are uploaded are now limited to those that have changed and that satisfy the search condition in the WHERE clause.

## Related Information

[UltraLite Clients \[page 73\]](#)

### 1.9.1.5.3 Dropping a Publication for UltraLite

Drop a publication for UltraLite.

#### Context

Dropping a table's publications allows you to synchronize all the tables and rows of that table in your UltraLite database.

You can drop a publication using either SQL Central or Interactive SQL.

#### Procedure

1. In SQL Central, connect to the UltraLite database.
2. In the left pane, double-click the *Publications* folder.
3. Right-click the publication and click *Delete*.
4. Click *Yes*.

#### Results

The publication is deleted.

#### Related Information

[UltraLite Clients \[page 73\]](#)

[DROP PUBLICATION Statement \[UltraLite\] \[page 546\]](#)

## 1.9.2 Microsoft ActiveSync Synchronization Overview

Synchronization through Microsoft ActiveSync can be summarized in a few steps.

1. Microsoft ActiveSync begins a synchronization session.
2. The Microsoft ActiveSync provider sends a synchronize notification message to the first registered application on the device. The application is started if it is not yet running.

3. WndProc is invoked for each registered application.
4. Once the application has determined that this is the synchronize notification message from Microsoft ActiveSync, the application calls `ULIsSynchronizeMessage` to invoke the database synchronization procedure.
5. Once synchronization is complete, the application calls `ULSignalSynclsComplete` to let the provider know that it has finished synchronizing.
6. Steps two-five are repeated for each application that has been registered with the provider.

## 1.9.3 UltraLite Synchronization Parameters

Synchronization parameters control the synchronization between an UltraLite database and the MobiLink server.

The way you set parameters depends on the specific UltraLite interface you are using.

### **i** Note

The parameters described only apply to UltraLite remote databases. Use the MobiLink SQL Anywhere client utility to synchronize SQL Anywhere remote databases.

## Required Parameters

The following parameters are required:

- Stream Type
- User Name
- Version

The synchronization function throws an exception, such as `SQLCode.SQLE_SYNC_INFO_INVALID` or its equivalent, if you do not set these parameters.

## Conflicting Parameters

You can specify at most one of these parameters:

- Download Only
- Ping
- Upload Only

The synchronization function throws an exception, such as `SQLCode.SQLE_SYNC_INFO_INVALID` or its equivalent, if you set more than one of these parameters to true.

**In this section:**

[Additional Parameters Synchronization Parameter \[page 95\]](#)

This synchronization parameter allows an application to supply additional parameters that are seldom used or that cannot be readily specified using other predefined parameters.

#### [Authentication Parameters Synchronization Parameter \[page 98\]](#)

Supplies parameters to authentication parameters in MobiLink events.

#### [Authentication Status Synchronization Parameter \[page 99\]](#)

This field is set by a synchronization to report the status of MobiLink user authentication. The MobiLink server provides this information to the client.

#### [Authentication Value Synchronization Parameter \[page 100\]](#)

This field is set by a synchronization to report results of a custom MobiLink user authentication script. The MobiLink server provides this information to the client.

#### [Download Only Synchronization Parameter \[page 101\]](#)

Prevents changes from being uploaded from the UltraLite database during this synchronization.

#### [Ignored Rows Synchronization Parameter \[page 102\]](#)

This field is set by a synchronization to indicate that rows were ignored by the MobiLink server during synchronization because of absent scripts.

#### [Keep Partial Download Synchronization Parameter \[page 103\]](#)

Controls whether UltraLite holds on to the partial download, rather than rolling back the changes, when a download fails because of a communications error during synchronization.

#### [New Password Synchronization Parameter \[page 104\]](#)

Sets a new MobiLink password associated with the user name.

#### [Number of Authentication Parameters Synchronization Parameter \[page 105\]](#)

Supplies the number of authentication parameters being passed to authentication parameters in MobiLink events.

#### [Observer Synchronization Parameter \[page 106\]](#)

Specifies a pointer to a callback function or event handler that monitors synchronization.

#### [Partial Download Retained Synchronization Parameter \[page 106\]](#)

Indicates whether UltraLite applied the changes that were downloaded rather than rolling back the changes when a download fails due to a communications error during synchronization.

#### [Password Synchronization Parameter \[page 107\]](#)

Specifies the MobiLink password associated with the user name.

#### [Ping Synchronization Parameter \[page 108\]](#)

Confirms communications between the UltraLite client and the MobiLink server. When this parameter is set to true, no synchronization takes place.

#### [Publications Synchronization Parameter \[page 109\]](#)

Specifies the publications to be synchronized.

#### [Resume Partial Download Synchronization Parameter \[page 110\]](#)

Resumes a failed download.

#### [Send Download Acknowledgement Synchronization Parameter \[page 111\]](#)

Instructs the MobiLink server that the client will provide a download acknowledgement.

#### [Stream Error Synchronization Parameter \[page 112\]](#)

Provides a structure to hold communications error reporting information.

#### [Stream Type Synchronization Parameter \[page 114\]](#)

Sets the MobiLink network protocol to use for synchronization.

#### [Stream Parameters Synchronization Parameter \[page 115\]](#)

Sets options to configure the network protocol.

#### [Sync Result Synchronization Parameter \[page 116\]](#)

Reports the status of a synchronization.

#### [Upload OK Synchronization Parameter \[page 117\]](#)

This field is set by a synchronization to report the status of data uploaded to the MobiLink server.

#### [Upload Only Synchronization Parameter \[page 118\]](#)

Indicates that there should be no downloads in the current synchronization, which can save communication time over slow communication links.

#### [User Data Synchronization Parameter \[page 119\]](#)

Makes application-specific information available to the synchronization observer.

#### [User Name Synchronization Parameter \[page 120\]](#)

Required. A string that the MobiLink server uses for authentication purposes.

#### [Version Synchronization Parameter \[page 121\]](#)

Defines the consolidated database version.

## Related Information

[MobiLink SQL Anywhere Client Utility \(dbmlsync\) Syntax](#)

### 1.9.3.1 Additional Parameters Synchronization Parameter

This synchronization parameter allows an application to supply additional parameters that are seldom used or that cannot be readily specified using other predefined parameters.

#### Syntax

The syntax varies depending on the API you use. The additional parameters are specified as a semicolon-delimited list of keyword=value pairs.

## Allowed Values

The following properties can be specified as part of the additional parameters setting:

Property name	Description
AllowDownloadDupRows	<p>Prevents errors from being raised when a synchronization encounters downloaded rows with duplicate primary keys.</p> <p>Set this property to 0 to raise errors and roll back the download; otherwise, set to 1 to raise warnings and continue the download.</p> <p>This property is only available in UltraLite C++.</p>
CheckpointStore	<p>Adds additional checkpoints of the database during synchronization to limit database growth during the synchronization process.</p> <p>Set this property to 1 to enable this feature, which is beneficial for large downloads with many updates but slows down synchronization; otherwise, set to 0, which is the default.</p>
DisableConcurrency	<p>Disallows database access from other threads during synchronization during the upload phase.</p> <p>Set this property to 0 to allow concurrent database access; otherwise, set to 1. By default, this property is set to 0.</p>



Property name	Description
TableOrder	<p>Sets the table order required for priority synchronization if the UltraLite default table ordering is not suitable for your deployment.</p> <p>Set this property to a list of table names, arranged in the desired order for upload. For UltraLite, use a comma-delimited list; for ulsync, use a semicolon-delimited list. By default, the order is based on foreign key relationships. Typically, the default is acceptable when the foreign keys on your consolidated database match the UltraLite remote database and there are no foreign key cycles.</p> <p>Quote tables names with either single or double quotes. For example, "Customer,Sales" and 'Customer,Sales' are both supported in UltraLite.</p> <p>If you include tables that are not included in the synchronization, they are ignored. Any tables that you do not list are appropriately sorted based on the foreign keys defined in the remote database.</p> <p>The order of tables on the download is the same as those you define for upload.</p> <p>You only need to explicitly set the table order if your UltraLite tables:</p> <ul style="list-style-type: none"> <li>• Are part of foreign key cycles. You must then list all tables that are part of a cycle.</li> <li>• Have different foreign key relationships in the consolidated database.</li> </ul>

## Example

UltraLite for C++ applications can set additional parameters as follows:

```
ul_sync_info info;
// ...
info.additional_parms = UL_TEXT(
    "AllowDownloadDupRows=1;
    CheckpointStore=1;
    DisableConcurrency=1;
    TableOrder=Customer,Sales"
);
```

UltraLiteJ applications can set additional parameters as follows:

```
SyncParms parms;
// ...
parms.setAdditionalParms(
    "AllowDownloadDupRows=1;CheckpointStore=1;DisableConcurrency=1" );
```

## 1.9.3.2 Authentication Parameters Synchronization Parameter

Supplies parameters to authentication parameters in MobiLink events.

### ☞ Syntax

The syntax varies depending on the API you use.

### Remarks

Parameters may be a user name and password, for example.

If you use this parameter, you must also supply the number of parameters.

### Allowed Values

An array of strings. Null is not allowed as a value for any of the strings, but you can supply an empty string.

### Example

UltraLite for C/C++ applications can set the parameters as follows:

```
ul_char * Params[ 3 ] = { UL_TEXT( "parm1" ),
                          UL_TEXT( "parm2" ),
                          UL_TEXT( "parm3" ) };
// ...
info.num_auth_parms = 3;
info.auth_parms = Params;
```

### Related Information

[Authentication Parameters](#)

[Number of Authentication Parameters Synchronization Parameter \[page 105\]](#)

[authenticate\\_parameters Connection Event](#)

## 1.9.3.3 Authentication Status Synchronization Parameter

This field is set by a synchronization to report the status of MobiLink user authentication. The MobiLink server provides this information to the client.

### ≡ Syntax

The syntax varies depending on the API you use.

## Allowed Values

The allowed values are held in an interface-specific enumeration. For example, for C/C++ applications the enumeration is as follows.

Constant	Value	Description
UL_AUTH_STATUS_UNKNOWN	0	Authorization status is unknown, possibly because the connection has not yet synchronized.
UL_AUTH_STATUS_VALID	1	User ID and password were valid at the time of synchronization.
UL_AUTH_STATUS_VALID_BUT_EXPIRES_SOON	2	User ID and password were valid at the time of synchronization but will expire soon.
UL_AUTH_STATUS_EXPIRED	3	Authorization failed: user ID or password have expired.
UL_AUTH_STATUS_INVALID	4	Authorization failed: bad user ID or password.
UL_AUTH_STATUS_IN_USE	5	Authorization failed: user ID is already in use.

## Remarks

If a custom `authenticate_user` synchronization script at the consolidated database returns a different value, the value is interpreted according to the rules given in an `authenticate_user` connection event.

If you are implementing a custom authentication scheme, the `authenticate_user` or `authenticate_user_hashed` synchronization script must return one of the allowed values of this parameter.

The parameter is set by the MobiLink server, and so is read-only.

## Example

UltraLite for C/C++ applications can access the parameter as follows:

```
ul_sync_info info;
// ...
returncode = info.auth_status;
```

## Related Information

[MobiLink Users in a Synchronization System](#)  
[authenticate\\_user Connection Event](#)

### 1.9.3.4 Authentication Value Synchronization Parameter

This field is set by a synchronization to report results of a custom MobiLink user authentication script. The MobiLink server provides this information to the client.

#### ⌘ Syntax

The syntax varies depending on the API you use. It is not available in UltraLiteJ.

## Remarks

The values set by the default MobiLink user authentication mechanism are described in the `authenticate_user` connection event and Authentication Status synchronization parameter.

The parameter is set by the MobiLink server, and so is read-only.

## Example

UltraLite for C/C++ applications can access the parameter as follows:

```
ul_sync_info info;
// ...
returncode = info.auth_value;
```

## Related Information

[authenticate\\_user](#) Connection Event

[authenticate\\_user\\_hashed](#) Connection Event

[Authentication Status Synchronization Parameter](#) [page 99]

### 1.9.3.5 Download Only Synchronization Parameter

Prevents changes from being uploaded from the UltraLite database during this synchronization.

☞ Syntax

The syntax varies depending on the API you use.

#### Default

False

#### Allowed Values

Boolean

#### Conflicts with

Ping and Upload Only

#### Remarks

Data changes are not uploaded when download-only synchronization occurs. However, information about the schema and the value stored in the progress counter is still uploaded. If the downloaded data conflicts with changes on the remote that have not been uploaded, then the synchronization fails and is rolled back.

## Example

The following example illustrates how to set the DownloadOnly synchronization parameter using the ulsync utility:

```
ulsync -c DBF=myuldb.udb
        "MobiLinkId=remoteA;ScriptVersion=2;DownloadOnly=ON;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
//...
info.download_only = ul_true;
```

## Related Information

[UltraLite Synchronization Client Features \[page 10\]](#)

[Upload Only Synchronization Parameter \[page 118\]](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

### 1.9.3.6 Ignored Rows Synchronization Parameter

This field is set by a synchronization to indicate that rows were ignored by the MobiLink server during synchronization because of absent scripts.

#### ☞ Syntax

The syntax varies depending on the API you use.

## Allowed Values

Boolean

## Remarks

The parameter is read-only.

## Example

UltraLite for C/C++ applications can access the parameter as follows:

```
ul_sync_info info;  
// ...  
res = info.ignored_rows;
```

### 1.9.3.7 Keep Partial Download Synchronization Parameter

Controls whether UltraLite holds on to the partial download, rather than rolling back the changes, when a download fails because of a communications error during synchronization.

#### ⚠ Syntax

The syntax varies depending on the API you use.

## Default

False, which indicates that UltraLite rolls back all changes after a failed download.

## Allowed Values

Boolean

## Example

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
// ...  
info.keep_partial_download = ul_true;
```

## Related Information

[Resumption of Failed Downloads](#)

[Resume Partial Download Synchronization Parameter \[page 110\]](#)

## 1.9.3.8 New Password Synchronization Parameter

Sets a new MobiLink password associated with the user name.

### ☞ Syntax

The syntax varies depending on the API you use. You can also set this parameter with `ulsync`.

### Allowed Values

String

### Remarks

The parameter is optional.

### Example

`ulsync` can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb  
"MobiLinkUid=remoteA;ScriptVersion=2;NewMobiLinkPwd=mynewpassword;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
//...  
info.new_password = UL_TEXT( "mlnewpass" );
```

### Related Information

[MobiLink Users in a Synchronization System](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)



## 1.9.3.9 Number of Authentication Parameters Synchronization Parameter

Supplies the number of authentication parameters being passed to authentication parameters in MobiLink events.

### ↳ Syntax

The syntax varies depending on the API you use. Not required for UltraLiteJ.

### Default

No parameters passed to a custom authentication script.

### Remarks

The parameter is used together with Authentication Parameters to supply information to custom authentication scripts.

### Example

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
//...  
info.num_auth_parms = 3;
```

### Related Information

[Authentication Parameters](#)

[Authentication Parameters Synchronization Parameter \[page 98\]](#)

[authenticate\\_parameters Connection Event](#)

## 1.9.3.10 Observer Synchronization Parameter

Specifies a pointer to a callback function or event handler that monitors synchronization.

The signature of the callback function that you need to implement to use is of the type `ul_sync_observer_fn`:

```
typedef void(UL_CALLBACK_FN *ul_sync_observer_fn) ( ul_sync_status * status );
```

### ☰ Syntax

The syntax varies depending on the API you use.

## Example

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
//...  
info.observer=callfunction;
```

## Related Information

[User Data Synchronization Parameter \[page 119\]](#)

## 1.9.3.11 Partial Download Retained Synchronization Parameter

Indicates whether UltraLite applied the changes that were downloaded rather than rolling back the changes when a download fails due to a communications error during synchronization.

This parameter is set by the synchronization.

### ☰ Syntax

The syntax varies depending on the API you use.

## Allowed Values

Boolean

## Remarks

The parameter is set during synchronization if a download error occurs and a partial download was retained. Partial downloads are retained only if Keep Partial Download is set to true.

## Example

Access the parameter as follows:

```
ul_sync_info info;  
//...  
returncode=info.partial_download_retained;
```

## Related Information

[Resumption of Failed Downloads](#)

[Keep Partial Download Synchronization Parameter \[page 103\]](#)

[Resume Partial Download Synchronization Parameter \[page 110\]](#)

## 1.9.3.12 Password Synchronization Parameter

Specifies the MobiLink password associated with the user name.

### ⚠ Syntax

The syntax varies depending on the API you use. You can also set this parameter with `ulsync`.

## Allowed Values

String

## Remarks

The parameter is optional.

This MobiLink user name and password are different than any database user ID and password, and serve to only identify and authenticate the application to the MobiLink server.

If the MobiLink client already has a password, use the New Password parameter to change it.

## Example

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb  
"MobiLinkUid=remoteA;ScriptVersion=2;MobiLinkPwd=mypassword;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
// ...  
info.password = UL_TEXT( "mypassword" );
```

## Related Information

[MobiLink Users in a Synchronization System](#)

[User Name Synchronization Parameter \[page 120\]](#)

[New Password Synchronization Parameter \[page 104\]](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

### 1.9.3.13 Ping Synchronization Parameter

Confirms communications between the UltraLite client and the MobiLink server. When this parameter is set to true, no synchronization takes place.

#### ☞ Syntax

The syntax varies depending on the API you use. You can also set this parameter with ulsync.

## Default

False

## Allowed Values

Boolean

## Remarks

When the MobiLink server receives a ping request, it connects to the consolidated database, authenticates the user, and then sends the authenticating user status and value back to the client.

If the ping succeeds, the MobiLink server issues an information message. If the ping does not succeed, it issues an error message.

If the MobiLink user ID cannot be found in the ml\_user system table and the MobiLink server is running with the command line option -zu+, the MobiLink server adds the user to ml\_user.

The MobiLink server may execute the following scripts, if they exist, for a ping request:

- begin\_connection
- authenticate\_user
- authenticate\_user\_hashed
- authenticate\_parameters
- end\_connection

## Example

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb  
"MobiLinkUid=remoteA;ScriptVersion=2;Ping=True;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
//...  
info.ping = ul_true;
```

## Related Information

[-pi dbmlsync Option](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

### 1.9.3.14 Publications Synchronization Parameter

Specifies the publications to be synchronized.

#### ≡ Syntax

The syntax varies depending on the API you use. You can also use this parameter with ulsync.

## Default

Synchronize all publications.

## Remarks

When synchronizing in C/C++, set the publications synchronization parameter to a **publication list**: a comma-separated list of publication names.

## Example

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb
"MobiLinkId=remoteA;ScriptVersion=2;Publications=UL_PUB_MYPUB1,UL_PUB_MYPUB2;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
// ...
info.publications = UL_TEXT( "Pubs1,Pubs3" );
```

## Related Information

[UltraLite Publications \[page 89\]](#)

[Publishing Data in UltraLite \[page 83\]](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

### 1.9.3.15 Resume Partial Download Synchronization Parameter

Resumes a failed download.

#### ☰ Syntax

The syntax varies depending on the API you use. You can also set this parameter with ulsync.

## Default

False

## Allowed Values

Boolean

## Remarks

The synchronization does not upload changes; it only downloads those changes that were to be downloaded in the failed download.

## Example

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
//...  
info.resume_partial_download = ul_true;
```

## Related Information

[Resumption of Failed Downloads](#)

[Keep Partial Download Synchronization Parameter \[page 103\]](#)

[Partial Download Retained Synchronization Parameter \[page 106\]](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

### 1.9.3.16 Send Download Acknowledgement Synchronization Parameter

Instructs the MobiLink server that the client will provide a download acknowledgement.

#### ⚙ Syntax

The syntax varies depending on the API you use. You can also set this parameter with `ulsync`.

## Default

False

## Allowed Values

Boolean

## Example

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb  
"MobiLinkId=remoteA;ScriptVersion=2;SendDownloadACK=true;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
//...  
info.send_download_ack = ul_true;
```

## Related Information

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

### 1.9.3.17 Stream Error Synchronization Parameter

Provides a structure to hold communications error reporting information.

#### ⌘ Syntax

The syntax varies depending on the API you use.

## Applies to

This parameter applies only to C++ interfaces.



## Allowed Values

The parameter has no default value, and must be explicitly set using one of the supported fields. The `ul_stream_error` fields are as follows:

### **stream\_error\_code**

For the error code suffixes, see `%SQLANY17%\SDK\Include\sserror.h`.

### **system\_error\_code**

A system-specific error code. For more information about the error code, you must look at your platform documentation. For Windows platforms, this is the Microsoft Developer Network documentation.

The following are common system errors on Windows:

#### **10048 (WSAADDRINUSE)**

Address already in use.

#### **10053 (WSAECONNABORTED)**

Software caused connection abort.

#### **10054 (WSAECONNRESET)**

The other side of the communication closed the socket.

#### **10060 (WSAETIMEDOUT)**

Connection timed out.

#### **10061 (WSAECONNREFUSED)**

Connection refused. Typically, the MobiLink server is not running or is not listening on the specified port.

### **error\_string**

A string with additional information, if available, for the `stream_error_code`. The string may or may not be empty. A non-empty `error_string` value provides information in addition to the `stream_error_code` value. For example, for a write error (error code 9) the error string is a number showing how many bytes it was trying to write.

## Remarks

UltraLite applications other than the UltraLite C++ Component receive communications error information as part of the Sync Result parameter.

The `stream_error` field is a structure of type `ul_stream_error`.

```
typedef struct {
    ss_error_code stream_error_code;
    asa_uint16    alignment;
    asa_int32    system_error_code;
    char         error_string[UL_STREAM_ERROR_STRING_SIZE];
} ul_stream_error, * p_ul_stream_error;
```

The structure is defined in `%SQLANY17%\SDK\Include\sserror.h`.

Check for a `SQLE_MOBILINK_COMMUNICATIONS_ERROR`:

```
ULConnection * conn;
ul_sync_info info;
...
conn->InitSynchInfo( &info );
if( !conn->Synchronize( &info ) ) {
    ULError const * error = conn->GetLastError();
    char buf[256];
    if( error->GetSQLCode() == SQLE_MOBILINK_COMMUNICATIONS_ERROR ) {
        error->GetString( buf, sizeof(buf) );
        printf( "%s\n", buf );
        // more handling for communication error
    }
}
```

## Related Information

[MobiLink Communication Error Messages](#)

[Sync Result Synchronization Parameter \[page 116\]](#)

### 1.9.3.18 Stream Type Synchronization Parameter

Sets the MobiLink network protocol to use for synchronization.

#### Syntax

The syntax varies depending on the API you use. You can also set this parameter with `ulsync`.

## Remarks

This parameter is required. It has no default value.

Most network protocols require protocol options to identify the MobiLink server address and other behavior. These options are supplied in the Stream Parameters parameter.

When the network protocol requires an option, pass that option using the Stream Parameters parameter; otherwise, set the Stream Parameters parameter to null.

The following stream types are available, but not all are available on all target platforms:

Network protocol	Description
HTTP	Synchronize over HTTP.

Network protocol	Description
HTTPS	Synchronize over HTTPS.  The HTTPS protocol uses TLS as its underlying security layer. It operates over TCP/IP.
TCP/IP	Synchronize over TCP/IP.
TLS	Synchronize over TCP/IP with Transport Layer Security (TLS). TLS secures client/server communications using digital certificates and public-key cryptography.

## Example

For UltraLite for C/C++ applications, set the parameter as follows:

```
Connection conn;
ul_sync_info info;
...
conn.InitSynchInfo( &info );
info.stream = "http";
```

## Related Information

[Transport Layer Security](#)

[UltraLite Network Protocol Options \[page 122\]](#)

[Supported Platforms](#)

[Stream Parameters Synchronization Parameter \[page 115\]](#)

[Certificate Creation Utility \(createcert\)](#)

[Certificate Viewer Utility \(viewcert\)](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

### 1.9.3.19 Stream Parameters Synchronization Parameter

Sets options to configure the network protocol.

#### ⌘ Syntax

The syntax varies depending on the API you use. You can also set this parameter with ulsync.

## Default

Null

## Allowed Values

String

## Remarks

This parameter is optional. It accepts a semicolon separated list of network protocol options. Each option is of the form `keyword=value`, where the allowed sets of keywords depends on the network protocol.

## Example

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
// ...  
info.stream_parms= UL_TEXT( "host=myserver;port=2439" );
```

## Related Information

[UltraLite Network Protocol Options \[page 122\]](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

### 1.9.3.20 Sync Result Synchronization Parameter

Reports the status of a synchronization.

#### ⚙ Syntax

The syntax varies depending on the API you use.

## Remarks

The parameter is set by UltraLite, and is read-only.

The C/C++ interface receives this information in separate parameters as part of a `ul_sync_info` struct. Otherwise, this information is defined as a compound parameter containing a variety of information in separate fields:

### Authentication Status

Reports success or failure of authentication.

### Ignored Rows

Reports the number of ignored rows.

### Stream Error information

The Stream Error information includes a Stream Error Code, Stream Error Context, Stream Error ID, and Stream Error System.

### Upload OK

Reports the success or failure of the upload phase.

## Related Information

[Authentication Status Synchronization Parameter \[page 99\]](#)

[Ignored Rows Synchronization Parameter \[page 102\]](#)

[Stream Error Synchronization Parameter \[page 112\]](#)

[Upload OK Synchronization Parameter \[page 117\]](#)

### 1.9.3.21 Upload OK Synchronization Parameter

This field is set by a synchronization to report the status of data uploaded to the MobiLink server.

#### ☰ Syntax

The syntax varies depending on the API you use.

## Remarks

The parameter is set by UltraLite, and so is read-only.

After synchronization, the parameter holds true if the upload was successful, and false otherwise. You can check this parameter if there was a synchronization error, to know whether data was successfully uploaded before the error occurred.

## Example

UltraLite for C/C++ applications can access the parameter as follows:

```
ul_sync_info info;  
//...  
returncode = info.upload_ok;
```

### 1.9.3.22 Upload Only Synchronization Parameter

Indicates that there should be no downloads in the current synchronization, which can save communication time over slow communication links.

#### ⚠ Syntax

The syntax varies depending on the API you use. You can also set this parameter with `ulsync`.

## Default

False

## Allowed Values

Boolean

## Conflicts with

Download Only, Ping, and Resume Partial Download

## Remarks

When set to true, the client waits for the upload acknowledgement from the MobiLink server, after which it terminates the synchronization session successfully.

## Example

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb
"MobiLinkId=remoteA;ScriptVersion=2;UploadOnly=True;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
// ...
info.upload_only = ul_true;
```

## Related Information

[UltraLite Client Synchronization Design \[page 79\]](#)

[Download Only Synchronization Parameter \[page 101\]](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

## 1.9.3.23 User Data Synchronization Parameter

Makes application-specific information available to the synchronization observer.

### Applies to

C/C++ applications only. Other components, such as UltraLite.NET, do not require a separate parameter to handle user data and so have no User Data parameter.

#### ⚡ Syntax

The syntax varies depending on the API you use.

### Remarks

When implementing the synchronization observer callback function or event handler, you can make application-specific information available by providing information using the User Data parameter.

## Related Information

[Observer Synchronization Parameter \[page 106\]](#)

### 1.9.3.24 User Name Synchronization Parameter

Required. A string that the MobiLink server uses for authentication purposes.

#### ☞ Syntax

The syntax varies depending on the API you use. You can also set this parameter with `ulsync`.

## Remarks

This parameter is required. Empty strings and NULL strings are universally rejected.

The parameter has no default value, and must be explicitly set.

The user name does not have to be unique when a remote ID is used.

This MobiLink user name and password are separate from any database user ID and password, and serves only to identify and authenticate the application to the MobiLink server.

For a user to be part of a synchronization system, you must register the user name with the MobiLink server. The user name is stored in the name column of the `ml_user` MobiLink system table in the consolidated database.

## Example

`ulsync` can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb "MobiLinkUid=remoteA;ScriptVersion=2;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
// ...  
info.user_name= UL_TEXT( "remoteA" );
```

## Related Information

[Remote IDs](#)



[MobiLink Users in a Synchronization System](#)  
[MobiLink Users in a Synchronization System](#)  
[Password Synchronization Parameter \[page 107\]](#)  
[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

## 1.9.3.25 Version Synchronization Parameter

Defines the consolidated database version.

### ☰ Syntax

The syntax varies depending on the API you use. You can also set this parameter with `ulsync`.

## Allowed Values

String

## Remarks

This parameter is required. Empty strings and NULL strings are universally rejected.

Each synchronization script in the consolidated database is marked with a version string. For example, there may be two different `download_cursor` scripts, identified by different version strings.

## Example

`ulsync` can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb "MobiLinkUid=remoteA;ScriptVersion=2;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;  
// ...  
info.version = UL_TEXT( "default" );
```

## Related Information

[Script Versions](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

## 1.9.4 UltraLite Network Protocol Options

You must set the network protocol in your application.

Each UltraLite database that synchronizes with a MobiLink server does so over a network protocol. Available network protocols include TCP/IP, HTTP, HTTPS, and TLS.

For the network protocol you set, you can choose from a set of corresponding protocol options to ensure that the UltraLite application can locate and properly communicate with the MobiLink server. The MobiLink client network protocol options provide information such as addressing information (host and port) and protocol-specific information.

### In this section:

[Synchronization Stream Options \[page 122\]](#)

You can provide the information needed to locate the MobiLink server in your application by setting the Stream Parameters parameter.

### Related Information

[Configuring UltraLite Clients to Use Transport Layer Security \[page 25\]](#)

[MobiLink Client Network Protocol Options](#)

[MobiLink Client Network Protocol Options](#)

[Stream Parameters Synchronization Parameter \[page 115\]](#)

[UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

### 1.9.4.1 Synchronization Stream Options

You can provide the information needed to locate the MobiLink server in your application by setting the Stream Parameters parameter.

### Related Information

[Stream Parameters Synchronization Parameter \[page 115\]](#)

## 1.10 UltraLite Deployment

In the majority of cases, development occurs on a Windows desktop or macOS with the final release target for UltraLite being the mobile device.

However, depending on your deployment environment, you can use various deployment mechanisms to install UltraLite.

UltraLite application projects may evolve with different iterations of the same UltraLite database: a development database, a test database, and a deployed production database. During the lifetime of a deployed database application, changes and improvements are first made in the development database, then propagated to the test database, before finally being distributed to the production database.

The modules you need to use for your UltraLite application depend on the platform you are targeting, the interface you are using, and the functionality you want to use.

### In this section:

#### [UltraLite Application Build and Deployment Specifications \[page 124\]](#)

There is a minimum set of requirements to build and deploy an UltraLite application for all supported platforms and devices, including the requirements for UltraLite database encryption.

#### [UltraLite Database Deployment Techniques \[page 130\]](#)

There are several techniques you can use to get the initial database file onto a device.

#### [Deploying UltraLite Database Schema Upgrades \[page 131\]](#)

Perform a schema upgrade.

#### [UltraLite Engine Startup \[page 133\]](#)

When using the UltraLite engine to manage data on a Microsoft Windows or Microsoft Windows Mobile device, your UltraLite application starts the engine automatically unless the application needs to explicitly provide the directory location of the engine.

#### [Registering Applications with the Microsoft ActiveSync Manager \[page 134\]](#)

Register applications that use Microsoft ActiveSync synchronization.

## Related Information

[UltraLite Data Management Components for Microsoft Windows Mobile \[page 21\]](#)

[How to Build and Deploy UltraLite C++ Applications \[page 667\]](#)

[UltraLite.NET Application Development \[page 600\]](#)

## 1.10.1 UltraLite Application Build and Deployment Specifications

There is a minimum set of requirements to build and deploy an UltraLite application for all supported platforms and devices, including the requirements for UltraLite database encryption.

### i Note

There may be versions of the UltraLite engine located in directories that contains the `_dev` suffix, such as the `x86_dev` directory. These versions contain development-time logging functionality that can be used to diagnose problems on platforms for debugging purposes. For production systems, use a version of the engine that is not in a `_dev` directory.

The following table provides the minimum requirements:

Platform or device	Minimum requirements	AES encryption requirements	FIPS 140-2 AES encryption requirements
Microsoft Windows Mobile and desktop (UltraLite C++ using static linkage)	Link against: <ul style="list-style-type: none"><li><code>ulrt.lib</code><sup>1</sup></li><li><code>ulbase.lib</code><sup>1</sup></li></ul>	Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.  Call the <code>EnableAesDBEncryption</code> method.	Set the DBKEY creation parameter to the encryption key when creating and connecting to the database.  Set the creation parameter <code>fips=yes</code> when creating the database.  Call the <code>EnableAesFipsDBEncryption</code> method.  Deploy: <ul style="list-style-type: none"><li><code>ulfips17.dll</code><sup>2</sup></li><li><code>libey32.dll</code><sup>2</sup></li><li><code>msvcr90.dll</code>/ <code>msvcr100.dll</code><sup>2</sup></li></ul>

Platform or device	Minimum requirements	AES encryption requirements	FIPS 140-2 AES encryption requirements
Microsoft Windows Mobile and desktop (UltraLite C++ using dynamic linkage)	<p>Link against:</p> <ul style="list-style-type: none"> <li><code>ulimp.lib</code><sup>1,10</sup></li> <li><code>ulbase.lib</code><sup>1</sup></li> </ul> <p>Deploy:</p> <ul style="list-style-type: none"> <li><code>ulrt17.dll</code><sup>1</sup></li> </ul>	<p>Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.</p> <p>Call the <code>EnableAesDBEncryption</code> method.</p>	<p>Set the DBKEY creation parameter to the encryption key when creating and connecting to the database.</p> <p>Set the creation parameter <code>fips=yes</code> when creating the database.</p> <p>Call the <code>EnableAesFipsDBEncryption</code> method.</p> <p>Deploy:</p> <ul style="list-style-type: none"> <li><code>ulfips17.dll</code><sup>2</sup></li> <li><code>libeay32.dll</code><sup>2</sup></li> <li><code>msvcr90.dll/</code> <code>msvcr100.dll</code><sup>2</sup></li> </ul>
Microsoft Windows Mobile and desktop (UltraLite C++ with the UltraLite engine)	<p>Link against:</p> <ul style="list-style-type: none"> <li><code>ulrtc.lib</code><sup>1</sup></li> <li><code>ulbase.lib</code><sup>1</sup></li> </ul> <p>Deploy:</p> <ul style="list-style-type: none"> <li><code>uleng17.exe</code><sup>2</sup></li> </ul>	<p>Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.</p>	<p>Set the DBKEY creation parameter to the encryption key when creating and connecting to the database.</p> <p>Set the creation parameter <code>fips=yes</code> when creating the database.</p> <p>Deploy:</p> <ul style="list-style-type: none"> <li><code>ulfips17.dll</code><sup>2</sup></li> <li><code>libeay32.dll</code><sup>2</sup></li> <li><code>msvcr90.dll/</code> <code>msvcr100.dll</code><sup>2</sup></li> </ul>
Microsoft Windows Mobile and desktop (UltraLite.NET)	<p>Add references to:</p> <ul style="list-style-type: none"> <li><a href="#">Sap.Data.UltraLite</a></li> <li><a href="#">Sap.Data.UltraLite.resources</a></li> </ul> <p>Deploy:</p> <ul style="list-style-type: none"> <li><code>Sap.Data.UltraLite.dll</code><sup>7</sup></li> <li><code>Sap.Data.UltraLite.resources.dll</code><sup>8</sup></li> <li><code>ulnet17.dll</code><sup>6</sup></li> </ul>	<p>Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.</p>	<p>Set the DBKEY creation parameter to the encryption key when creating and connecting to the database.</p> <p>Set the creation parameter <code>fips=yes</code> when creating the database.</p> <p>Deploy:</p> <ul style="list-style-type: none"> <li><code>ulfips17.dll</code><sup>2</sup></li> <li><code>libeay32.dll</code><sup>2</sup></li> <li><code>msvcr90.dll/</code> <code>msvcr100.dll</code><sup>2</sup></li> </ul>

Platform or device	Minimum requirements	AES encryption requirements	FIPS 140-2 AES encryption requirements
Microsoft Windows Mobile and desktop (UltraLite.NET with the UltraLite engine)	<p>Add references to:</p> <ul style="list-style-type: none"> <li><a href="#">Sap.Data.UltraLite</a></li> <li><a href="#">Sap.Data.UltraLite.resources</a></li> </ul> <p>Deploy:</p> <ul style="list-style-type: none"> <li><code>Sap.Data.UltraLite.dll</code><sup>7</sup></li> <li><code>Sap.Data.UltraLite.resources.dll</code><sup>8</sup></li> <li><code>ulnetclient17.dll</code><sup>6</sup></li> <li><code>uleng17.exe</code><sup>2</sup></li> </ul>	<p>Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.</p>	<p>Set the DBKEY creation parameter to the encryption key when creating and connecting to the database.</p> <p>Set the creation parameter <code>fips=yes</code> when creating the database.</p> <p>Deploy:</p> <ul style="list-style-type: none"> <li><code>ulfips17.dll</code><sup>2</sup></li> <li><code>libeay32.dll</code><sup>2</sup></li> <li><code>msvcr90.dll/msvcr100.dll</code><sup>2</sup></li> </ul>
macOS and iOS (UltraLite C++)	<p>Add to your Xcode project:</p> <ul style="list-style-type: none"> <li><code>libulrt.a</code><sup>9</sup></li> <li><code>libulbase.a</code><sup>9</sup> (macOS only)</li> </ul>	<p>Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.</p> <p>Call the <code>EnableAesDBEncryption</code> method.</p>	Not applicable
Linux (UltraLite C++)	<p>Link against:</p> <ul style="list-style-type: none"> <li><code>libulrt.a</code><sup>3</sup></li> <li><code>libulbase.a</code><sup>3</sup></li> </ul>	<p>Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.</p> <p>Call the <code>EnableAesDBEncryption</code> method.</p>	Not applicable
Microsoft Windows Phone (UltraLite for Microsoft Windows Phone)	<p>Add <code>UltraLite.winmd</code><sup>13</sup> to your Microsoft Windows Phone project.</p>	<p>Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.</p>	Not applicable
Android (UltraLiteJ)	<p>Add to your Android project:</p> <ul style="list-style-type: none"> <li><code>UltraLiteJNI17.jar</code><sup>5</sup></li> <li><code>libultralitej17.so</code><sup>4</sup></li> </ul>	<p>Use the DBKEY creation parameter or the <code>setEncryptionKey</code> method to set the encryption key when creating or connecting to the database.</p> <p>Call the <code>EnableAesDBEncryption</code> method.</p>	Not applicable

<sup>1</sup> For Microsoft Windows Mobile, this file is located in %SQLANY17%\UltraLite\CE\Arm.50\Lib. For Microsoft Windows, it is located in %SQLANY17%\UltraLite\Windows\x64\Lib\VS9 or %SQLANY17%\UltraLite\Windows\x86\Lib\VS9.

<sup>2</sup> FIPS is not supported on Microsoft Windows Mobile. On Microsoft Windows, these files are located in %SQLANY17%\UltraLite\Windows\x64 or %SQLANY17%\UltraLite\Windows\x86.

<sup>3</sup> This file is located in /opt/sqlanywhere17/ultralite/linux/x64/lib.

<sup>4</sup> This file is located in %SQLANY17%\UltraLite\UltraLiteJ\Android\ARM.

<sup>5</sup> This file is located in %SQLANY17%\UltraLite\UltraLiteJ\Android.

<sup>6</sup> For Microsoft Windows Mobile, this file is located in %SQLANY17%\UltraLite\UltraLite.NET\CE\Arm.50. For Microsoft Windows, it is located in %SQLANY17%\UltraLite\UltraLite.NET\x64 or %SQLANY17%\UltraLite\UltraLite.NET\win32.

<sup>7</sup> This file is located in %SQLANY17%\UltraLite\UltraLite.NET\Assembly\V2.

<sup>8</sup> This file is located in %SQLANY17%\UltraLite\UltraLite.NET\Assembly\V2\en.

<sup>9</sup> For macOS, this file is located in /Applications/SQLAnywhere17/System/ultralite/macosx/x86\_64. For iOS, UltraLite runtimes must be built after installation. Follow the instructions provided in install-dir/ultralite/iphone/readme.txt.

<sup>10</sup> When linking against this library, define the UL\_USE\_DLL preprocessor macro when compiling. For example, specify the following:

```
-DUL_USE_DLL
```

<sup>11</sup> Required for over-the-air (OTA) deployment only. Alternatively, you can create your own .jad file that deploys UltraLiteJ with your application.

<sup>12</sup> The WinRT components for the ARM, x86, and x64 processors are located in the UltraLite\UWP\Windows\8.0\ and UltraLite\UWP\WindowsPhone\8.0\ directories of your SQL Anywhere installation. The Microsoft Windows Phone 8 emulator for x86 processors is included in the respective directory.

## Additional build and deployment requirements for synchronization and compression

The following table describes the stream, protocol option, and code requirements for building and deploying an UltraLite application that uses synchronization:

### i Note

The HTTPS stream option can be enabled in the UltraLiteJ API by passing the SyncParms.HTTPS\_STREAM constant to the Connection.createSyncParms method

Synchronization type	Stream option specification	Protocol option requirements	Method call requirements for UltraLite C++
TCP/IP	"tcpip"	None	<ul style="list-style-type: none"> <li>EnableTcpipSynchronization</li> </ul>
HTTP	"http"	None	<ul style="list-style-type: none"> <li>EnableHttpSynchronization</li> </ul>
RSA TLS	"tls"	None	<ul style="list-style-type: none"> <li>EnableTlsSynchronization</li> <li>EnableRsaSyncEncryption</li> </ul>
RSA HTTPS	"https"	None	<ul style="list-style-type: none"> <li>EnableHttpsSynchronization</li> <li>EnableRsaSyncEncryption</li> </ul>
FIPS 140-2 RSA TLS	"tls"	<i>fips=yes</i>	<ul style="list-style-type: none"> <li>EnableTlsSynchronization</li> <li>EnableRsaFipsSyncEncryption</li> </ul>
FIPS 140-2 RSA HTTPS	"https"	<i>fips=yes</i>	<ul style="list-style-type: none"> <li>EnableHttpsSynchronization</li> <li>EnableRsaFipsSyncEncryption</li> </ul>

The following table describes additional protocol option and code requirements for building and deploying an UltraLite application that uses compression or end-to-end encryption:

Compression and stream encryption options	Protocol option requirements	Method call requirements for UltraLite C++ and UltraLiteJ
ZLIB compression	<ul style="list-style-type: none"> <li><i>compression=zlib</i></li> </ul>	<ul style="list-style-type: none"> <li><b>C++:</b> EnableZlibSyncCompression</li> <li><b>Java:</b> setZlibCompression</li> </ul>
RSA E2EE	<ul style="list-style-type: none"> <li><i>e2ee_public_key= key-file</i></li> </ul>	<ul style="list-style-type: none"> <li><b>C++:</b> EnableRsaE2ee</li> <li><b>Java:</b> setE2eePublicKey</li> </ul>
FIPS 140-2 RSA E2EE	<ul style="list-style-type: none"> <li><i>e2ee_public_key= key-file</i></li> <li><i>fips=yes</i></li> </ul>	<ul style="list-style-type: none"> <li><b>C++:</b> EnableRsaFipsE2ee</li> <li><b>Java:</b> Not applicable</li> </ul>

The following table illustrates additional build and deployment requirements for compression and encrypted synchronization:

### Note

There are no additional build and deployment requirements for TCP/IP and HTTP synchronization.



Platform or device	ZLIB compression requirements	RSA TLS, RSA HTTPS, and RSA E2EE requirements	FIPS 140-2 RSA TLS, FIPS 140-2 RSA HTTPS, and FIPS 140-2 RSA E2EE requirements
Microsoft Windows Mobile and desktop (UltraLite C++ using static linkage)	None	Link against: <ul style="list-style-type: none"> <li>ulrsa.lib<sup>1</sup></li> </ul>	Deploy: <ul style="list-style-type: none"> <li>mlcrsafips17.dll<sup>2</sup></li> <li>libeay32.dll<sup>2</sup></li> <li>ssleay32.dll<sup>2</sup></li> <li>msvcr90.dll/ msvcr100.dll<sup>2</sup></li> </ul>
Microsoft Windows Mobile and desktop (UltraLite C++ using dynamic linkage)	Deploy: <ul style="list-style-type: none"> <li>mlczlib17.dll<sup>2</sup></li> </ul>	Deploy: <ul style="list-style-type: none"> <li>mlcrsa17.dll<sup>2</sup></li> </ul>	Deploy: <ul style="list-style-type: none"> <li>mlcrsafips17.dll<sup>2</sup></li> <li>libeay32.dll<sup>2</sup></li> <li>ssleay32.dll<sup>2</sup></li> <li>msvcr90.dll/ msvcr100.dll<sup>2</sup></li> </ul>
Microsoft Windows Mobile and desktop (UltraLite C++ with the UltraLite engine)	Deploy: <ul style="list-style-type: none"> <li>mlczlib17.dll<sup>2</sup></li> </ul>	Deploy: <ul style="list-style-type: none"> <li>mlcrsa17.dll<sup>2</sup></li> </ul>	Deploy: <ul style="list-style-type: none"> <li>mlcrsafips17.dll<sup>2</sup></li> <li>libeay32.dll<sup>2</sup></li> <li>ssleay32.dll<sup>2</sup></li> <li>msvcr90.dll/ msvcr100.dll<sup>2</sup></li> </ul>
Microsoft Windows Mobile and desktop (UltraLite.NET)	Deploy: <ul style="list-style-type: none"> <li>mlczlib17.dll<sup>2</sup></li> </ul>	Deploy: <ul style="list-style-type: none"> <li>mlcrsa17.dll<sup>2</sup></li> </ul>	Deploy: <ul style="list-style-type: none"> <li>mlcrsafips17.dll<sup>2</sup></li> <li>libeay32.dll<sup>2</sup></li> <li>ssleay32.dll<sup>2</sup></li> <li>msvcr90.dll/ msvcr100.dll<sup>2</sup></li> </ul>
Microsoft Windows Mobile and desktop (UltraLite.NET with the UltraLite engine)	Deploy: <ul style="list-style-type: none"> <li>mlczlib17.dll<sup>2</sup></li> </ul>	Deploy: <ul style="list-style-type: none"> <li>mlcrsa17.dll<sup>2</sup></li> </ul>	Deploy: <ul style="list-style-type: none"> <li>mlcrsafips17.dll<sup>2</sup></li> <li>libeay32.dll<sup>2</sup></li> <li>ssleay32.dll<sup>2</sup></li> <li>msvcr90.dll/ msvcr100.dll<sup>2</sup></li> </ul>
macOS and iOS (UltraLite C++)	None	None	Not applicable
Linux (UltraLite C++)	None	Link against: <ul style="list-style-type: none"> <li>libulrsa.a<sup>3</sup></li> </ul>	Not applicable

Platform or device	ZLIB compression requirements	RSA TLS, RSA HTTPS, and RSA E2EE requirements	FIPS 140-2 RSA TLS, FIPS 140-2 RSA HTTPS, and FIPS 140-2 RSA E2EE requirements
Microsoft Windows Phone (UltraLite for Microsoft Windows Phone)	None	Not applicable	Not applicable
Android (UltraLiteJ)	None	Deploy: <ul style="list-style-type: none"> <li>libmlcrsa17.so<sup>4</sup></li> </ul>	Not applicable

<sup>1</sup> For Microsoft Windows Mobile, this file is located in %SQLANY17%\UltraLite\CE\Arm.50\Lib. For Microsoft Windows, it is located in %SQLANY17%\UltraLite\Windows\x64\Lib\VS9 or %SQLANY17%\UltraLite\Windows\x86\Lib\VS9.

<sup>2</sup> FIPS is not supported on Microsoft Windows Mobile. On Microsoft Windows, these files are located in %SQLANY17%\UltraLite\Windows\x64 or %SQLANY17%\UltraLite\Windows\x86.

<sup>3</sup> This file is located in /opt/sqlanywhere17/ultralite/linux/x64/lib.

<sup>4</sup> This file is located in %SQLANY17%\UltraLite\UltraLiteJ\Android\ARM.

## Related Information

[How to Build and Deploy UltraLite C++ Applications \[page 667\]](#)

[How to Deploy UltraLite.NET Applications \[page 621\]](#)

[MobiLink Client Network Protocol Options](#)

## 1.10.2 UltraLite Database Deployment Techniques

There are several techniques you can use to get the initial database file onto a device.

- Use the UltraLite API in your application to create the initial database file.
- Create a schema file from a SQL script and use the ALTER DATABASE SCHEMA FROM FILE statement.
- Use UltraLite FileTransfer methods to download the initial database file if it does not already exist on the device.
- Bundle the initial database with the application.
- When deploying to a Microsoft Windows or Microsoft Windows Mobile device, use central administration to send down the initial UDB file, or send a command to create the initial database file and execute SQL to give it schema.

## Related Information

[Manage Remote Databases](#)

[Deploying UltraLite Database Schema Upgrades \[page 131\]](#)

[ALTER DATABASE SCHEMA FROM FILE Statement \[UltraLite\] \[page 520\]](#)

### 1.10.3 Deploying UltraLite Database Schema Upgrades

Perform a schema upgrade.

#### Prerequisites

The SQL file you use must contain the entire new schema.

#### Context

UltraLite database schema upgrades can be deployed using one of the following techniques:

##### Individual DDL statements

For example, execute the following statement to create a new publication:

```
dbconnection->ExecuteStatement("CREATE PUBLICATION p (table t)");
```

##### The ALTER DATABASE SCHEMA FROM FILE statement

This statement can be used to perform schema upgrades when you do not know the DDL statement requirements, or do not want to specify the individual DDL statements.

##### Caution

Do not reset a device during a schema upgrade. If you reset the device during a schema upgrade, data will be lost and the UltraLite database marked as "bad."

UltraLite executes the following steps when you upgrade an UltraLite database schema with the ALTER DATABASE SCHEMA FROM FILE statement:

1. Both the new and existing database schemas are compared to see what differs.
2. The schema of the existing database is altered.
3. Rows that do not fit the new schema are dropped. For example:
  - If you add a uniqueness constraint to a table and there are multiple rows with the same values, all but one row will be dropped.
  - If you try to change a column domain and a conversion error occurs, then that row is dropped. For example, if you have a VARCHAR column and convert it to an INT column and a row has the value ABCD, then that row is dropped.

- If your new schema has new foreign keys where the foreign row does not have a matching primary row, these rows are dropped.
4. When rows are dropped, a SQLE\_ROW\_DROPPED\_DURING\_SCHEMA\_UPGRADE (130) warning is raised.

## Procedure

1. Create a SQL script of DDL statements to create a completely new schema.

You can keep a master schema on your computer and update the schema as your application changes.

Use either the ulinit or ulunload utilities to extract the DDL statements required for your script. By using these utilities with the following options, you ensure that the DDL statements are syntactically correct:

- For an UltraLite database, use the ulunload utility with the `-n` and `-s [ schema-file ]` options. For example:

```
ulunload -c dbf=mydatabase.udb -n -s MySchema.sql
```

- For a SQL Anywhere database, use the ulinit utility with the `-a`, `-l [ schema-file ]`, and `-n [ publication-name ]` option. For example:

```
ulinit -a "dsn=mysqlanywheredatabase" -l MySchema.sql -n MyPub Temp.udb
```

If you do not use the ulunload or ulinit utilities, review the script and ensure the following:

- The script declares the entire desired schema with CREATE statements.
  - Tables, columns, and publications are not renamed. The RENAME operation is not supported. Renamed tables are processed as a DROP TABLE and CREATE TABLE operation.
  - There are no non-DDL statements, including non-DDL statements that may not have the effect you expect.
  - Words in the SQL statement are separated by spaces.
  - Only one SQL statement appears in each line.
  - Comments are prepended with double hyphens (-), and only occur at the start of a line.
  - Each statement is separated by a line containing exactly the word `GO`.
2. Deploy the new SQL script file.
  3. Ensure that the database is synchronized.
  4. Run the new statement on the device. For example:

```
ALTER DATABASE SCHEMA FROM FILE 'MySchema.sql'
```

## Results

The schema is updated.

## Related Information

[UltraLite Database Schemas \[page 52\]](#)

[ALTER DATABASE SCHEMA FROM FILE Statement \[UltraLite\] \[page 520\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Database Unload Utility \(ulunload\) \[page 243\]](#)

### 1.10.4 UltraLite Engine Startup

When using the UltraLite engine to manage data on a Microsoft Windows or Microsoft Windows Mobile device, your UltraLite application starts the engine automatically unless the application needs to explicitly provide the directory location of the engine.

When an UltraLite application attempts to start the UltraLite engine, the application searches the following directories:

Client platform	Directory locations
Microsoft Windows desktop	<ol style="list-style-type: none"><li>1. The directory of the application that is starting it</li><li>2. The current working directory</li><li>3. The system path</li><li>4. The SQL Anywhere install directory (either under <code>bin32</code> or <code>bin64</code>), depending on whether the client is 32-bit or 64-bit</li></ol>
Microsoft Windows Mobile/CE	<ol style="list-style-type: none"><li>1. <code>\Windows\</code></li><li>2. <code>\</code> (the root directory)</li><li>3. <code>\UltraLiteDB\</code></li></ol>
Linux	<ol style="list-style-type: none"><li>1. The directory of the application that is auto-starting it</li><li>2. <code>%SQLANY17%/bin32</code></li></ol>

If the UltraLite engine is stored in a different location, start the engine by specifying the START connection parameter.

For example, a connection string to the database or connection code for a Microsoft Windows Mobile client application might use the following START parameter value:

```
"START=\Program Files\MyApp\uleng17.exe"
```

## Related Information

[UltraLite Application Build and Deployment Specifications \[page 124\]](#)

[UltraLite START Connection Parameter \[page 200\]](#)

## 1.10.5 Registering Applications with the Microsoft ActiveSync Manager

Register applications that use Microsoft ActiveSync synchronization.

### Context

You can register your application for use with Microsoft ActiveSync either by using the ActiveSync Provider Installation utility or using the Microsoft ActiveSync Manager itself.

The following task describes how to use the Microsoft ActiveSync Manager to register your application:

### Procedure

1. Launch Microsoft ActiveSync.
2. From the Microsoft ActiveSync window, click *Options*.
3. From the list of information types, click *MobiLink Clients* and click *Settings*.
4. In the *MobiLink Synchronization* window, click *New*.
5. Enter the following information for your application:

#### **Application name**

A name identifying the application that appears in the Microsoft ActiveSync user interface.

#### **Class name**

The registered class name for the application.

#### **Path**

The location of the application on the device.

#### **Arguments**

Any command line arguments to be used when Microsoft ActiveSync starts the application.

6. Click *OK* to register the application.

### Results

The application is registered with Microsoft ActiveSync.

### Next Steps

Copy the application to the device.

## Related Information

[Assigning Class Names for Applications \[page 711\]](#)

# 1.11 Tutorial: Building the UltraLite CustDB Sample Application

In this tutorial you learn how to run the MobiLink server to carry out data synchronization between the consolidated database and the UltraLite remote, use SQL Central to browse the data in the UltraLite remote, and manage UltraLite databases with UltraLite utilities.

## Context

Different versions of the application code exist for each supported programming interface and platform. However, this tutorial references the compiled version of the application for Windows desktops only. Each version varies to conform to the conventions of each platform.

1. [Lesson 1: Building and Running the CustDB Application \[page 136\]](#)  
Build the CustDB application.
2. [Lesson 2: Starting the MobiLink Server and Performing an Initial Synchronization \[page 137\]](#)  
Start the MobiLink server and synchronize the CustDB database with the UltraLite database using the CustDB application.
3. [Lesson 3: Updating Data in the UltraLite Database \[page 138\]](#)  
Use the CustDB application to add, update, and delete data in the remote database.
4. [Lesson 4: Synchronizing the UltraLite Database with the Consolidated Database \[page 140\]](#)  
Synchronize databases and use either Interactive SQL or SQL Central to connect to the consolidated database and confirm that your changes were synchronized.
5. [Lesson 5: Browsing MobiLink Synchronization Scripts \[page 141\]](#)  
Browse synchronization scripts to get a better understanding of how the CustDB synchronization logic works.

## Related Information

[CustDB Sample Application Overview \[page 17\]](#)

[CustDB Sample for MobiLink](#)

[Users in the CustDB Sample](#)

[Tables in the CustDB Databases](#)

## 1.11.1 Lesson 1: Building and Running the CustDB Application

Build the CustDB application.

### Procedure

1. For non-Windows environments, build the CustDB application.
  - a. Open a CustDB project file in the appropriate environment.
  - b. Compile the source code.
2. Copy the CustDB application.

For Windows 32-bit environments, copy the CustDB application, `%SQLANY17%\UltraLite\Windows\x86\custdb.exe`, to the `%SQLANYSAMP17%\UltraLite\CustDB` directory.

### Results

The CustDB application is compiled.

#### **i** Note

If the Mobilink server is not running, starting the CustDB application displays an error message because it is unable to synchronize.

### Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building the UltraLite CustDB Sample Application \[page 135\]](#)

**Next task:** [Lesson 2: Starting the MobiLink Server and Performing an Initial Synchronization \[page 137\]](#)

### Related Information

[CustDB File Locations for UltraLite \[page 19\]](#)



## 1.11.2 Lesson 2: Starting the MobiLink Server and Performing an Initial Synchronization

Start the MobiLink server and synchronize the CustDB database with the UltraLite database using the CustDB application.

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

### Procedure

1. Click **Start > Programs > SQL Anywhere 17 > MobiLink > Synchronization Server Sample**. Or, run the following command:

```
mllsrv17 -c "DSN=SQL Anywhere 17 CustDB;uid=ml_server;pwd=sql" -vcrs
```

Use `mobmlink.sh` on macOS or Linux. You must specify the password for the sample database. For example:

```
mobmlink.sh sql
```

The window displays messages about the MobiLink server's status.

2. Run the CustDB application. In Windows, run `%SQLANYSAMPI7%\UltraLite\CustDB\custdb.exe`.
3. On the *File* menu, click *Synchronize*.

The application synchronizes and the MobiLink server messages window displays messages showing the synchronization taking place.

The synchronization script determines which subset of customers, products, and orders is downloaded to the application when user 50 logs in. In this case, only orders that have not yet been approved are downloaded.

4. Confirm that the company name and a sample order appear in the application window.

### Results

The CustDB application synchronizes with the consolidated database.

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building the UltraLite CustDB Sample Application \[page 135\]](#)

**Previous task:** [Lesson 1: Building and Running the CustDB Application \[page 136\]](#)

**Next task:** [Lesson 3: Updating Data in the UltraLite Database \[page 138\]](#)

## 1.11.3 Lesson 3: Updating Data in the UltraLite Database

Use the CustDB application to add, update, and delete data in the remote database.

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

### Procedure

1. Browse orders.

Browsing orders is accomplished by using a similar method for each version of the CustDB application. By browsing an order, you are scrolling through the data in your local UltraLite database. Because customers are sorted alphabetically, you can easily scroll through the list and locate a customer by name.

- a. To scroll down the list of customers, click *Next*.
- b. To scroll up through the list of customers, click *Previous*.

2. Add an order.

Adding an order is carried out in a similar way in each version of the CustDB application. By adding an order, you modify the data in your local UltraLite database. This data is not shared with the consolidated database until you synchronize.

- a. Click **► Order ► New ▾**.
- b. In the *Customer* list, use the directional keys to scroll down and click **Basements R Us**.
- c. In the *Product* list, use the directional keys to scroll down and click **Screwmaster Drill**. The price of this item is automatically entered in the *Price* field.
- d. In the *Quantity* field, type **20**.
- e. In the *Discount* field, type **5** (percent) and click *OK*.

### 3. Approve, deny, and delete orders.

Because you have authenticated your identity as user ID 50, you are a manager that can perform all the same tasks as a sales person, but you have the added ability to accept or reject orders. By accepting or rejecting an order, you change its status and add an additional note for the sales person to review. However, the data in the consolidated database is unchanged until you synchronize.

- a. Approve the order for **Apple Street Builders**.
  1. To locate the customer, click *Previous*.
  2. To approve the order, click *Order* and then *Approve*.
  3. In the *Note* list, click **Good** and then click *OK*.  
The order appears with a status of *Approved*.
- b. Deny the order for **Art's Renovations**.
  1. Go to the next order in the list, which is from **Art's Renovations**.
  2. To deny the order, click *Order* and then *Deny*.
  3. In the *Note* list, click *Discount Is Too High* and then click *OK*.  
The order appears with a status of *Denied*.
- c. Delete the order for **Awnings R Us**.
  1. Go to the next order in the list, which is from **Awnings R Us**.
  2. Delete this order by choosing **Order > Delete**.
  3. Click *OK* to confirm the deletion.  
The order is marked as deleted. However, the current data remains in the UltraLite remote database until you synchronize changes to the consolidated database.

## Results

Modifications to the data in the UltraLite database are saved but not synchronized with the CustDB database.

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building the UltraLite CustDB Sample Application \[page 135\]](#)

**Previous task:** [Lesson 2: Starting the MobiLink Server and Performing an Initial Synchronization \[page 137\]](#)

**Next task:** [Lesson 4: Synchronizing the UltraLite Database with the Consolidated Database \[page 140\]](#)

## Related Information

[Tables in the CustDB Databases](#)

## 1.11.4 Lesson 4: Synchronizing the UltraLite Database with the Consolidated Database

Synchronize databases and use either Interactive SQL or SQL Central to connect to the consolidated database and confirm that your changes were synchronized.

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

### Context

The synchronization process for the CustDB application removes approved orders from your database.

### Procedure

1. Synchronize the UltraLite database.

From the *File* menu, click *Synchronize Database*.

2. Confirm that synchronization took place.

At the remote database, you can confirm that all required transactions occurred by checking that the order for **Awnings R Us** is now deleted. Perform this action by browsing the orders to confirm the absence of this entry.

At the consolidated database, you can also confirm that all required actions occurred by checking data.

- Confirm that synchronization took place by using SQL Central.
  1. Click **Start** > **Programs** > **SQL Anywhere 17** > **Administration Tools** > **SQL Central**.
  2. Click **Connections** > **Connect With SQL Anywhere 17**.
  3. In the *Action* dropdown menu, click *Connect With An ODBC Data Source*.
  4. Click *ODBC Data Source Name*.
  5. Click *Browse* and click *SQL Anywhere 17 CustDB*.
  6. Click *OK*.
  7. Click *Connect*.
  8. Double-click *Tables*.
  9. Double-click **ULOrder**.
  10. Click the *Data* tab and verify that order 5100 is approved, order 5101 is denied, and order 5102 is deleted.
- Confirm that synchronization took place using Interactive SQL.

1. Connect to the consolidated database from Interactive SQL.
  1. Click [Start](#) [Programs](#) [SQL Anywhere 17](#) [Administration Tools](#) [Interactive SQL](#).
  2. In the *Action* dropdown list, click [Connect With An ODBC Data Source](#).
  3. Click [ODBC Data Source Name](#) and click [SQL Anywhere 17 CustDB](#).
  4. Click [Connect](#).
2. To confirm that the approval and denial have been synchronized, execute the following statement:

```
SELECT order_id, status
FROM ULOrder
WHERE status IS NOT NULL;
```

The results show that order 5100 is approved and 5101 is denied.

3. The deleted order has an order\_id of 5102. The following query returns no rows, demonstrating that the order has been removed from the system:

```
SELECT *
FROM ULOrder
WHERE order_id = 5102
```

## Results

The approved orders are removed from the database and you confirmed the removal.

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building the UltraLite CustDB Sample Application \[page 135\]](#)

**Previous task:** [Lesson 3: Updating Data in the UltraLite Database \[page 138\]](#)

**Next task:** [Lesson 5: Browsing MobiLink Synchronization Scripts \[page 141\]](#)

## 1.11.5 Lesson 5: Browsing MobiLink Synchronization Scripts

Browse synchronization scripts to get a better understanding of how the CustDB synchronization logic works.

## Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

## Procedure

1. Open *SQL Central*.

Click ► *Start* ► *Programs* ► *SQL Anywhere 17* ► *Administration Tools* ► *SQL Central* ►.

2. Connect to the consolidated database.

1. In the *Context* field, choose *SQL Central* and then double-click *MobiLink 17*.
2. In the right pane of *SQL Central*, double-click *CustDB* and then double-click *Consolidated Databases*.
3. Right-click *CustDB* and then click *Connect*.
4. In the *Action* dropdown menu, click *Connect With An ODBC Data Source*.
5. Click *ODBC Data Source Name*.
6. Click *Browse* and click *SQL Anywhere 17 CustDB*.
7. Click *OK*.
8. Click *Connect*.

3. Set up the MobiLink system.

Right-click *CustDB*, click *Check MobiLink System Setup*, and then click **OK**.

4. Click ► *View* ► *Folders* ► if the *Folders* option is not already selected.
5. In the left pane, expand ► *Consolidated Databases* ► *CustDB* ►.
6. Click *Connection Scripts*.

The right pane lists a set of synchronization scripts and a set of events with which these scripts are associated. As the MobiLink server carries out the synchronization process, it triggers a sequence of events. Any synchronization script associated with an event is run at that time. By writing synchronization scripts and assigning them to synchronization events, you can control the actions that are carried out during synchronization.

7. Click *Synchronized Tables*.

A set of scripts specific to this table, and their corresponding events appears. These scripts control the way that data is synchronized with the remote databases.

## Results

You have reviewed the synchronization scripts.

**Task overview:** [Tutorial: Building the UltraLite CustDB Sample Application \[page 135\]](#)

**Previous task:** [Lesson 4: Synchronizing the UltraLite Database with the Consolidated Database \[page 140\]](#)

## Related Information

[Synchronization Scripts](#)  
[Synchronization Logic Source Code](#)  
[CustDB Sample for MobiLink](#)  
[UltraLite Clients \[page 73\]](#)  
[Connection Scripts](#)  
[Table Scripts](#)

## 1.12 UltraLite Database Reference

UltraLite provides many tools and features to help you run, manage, and configure UltraLite databases.

### In this section:

#### [UltraLite Options \[page 143\]](#)

There are several options you can control when creating your UltraLite database. These are designed to help with the wide variety of UltraLite uses. Most options specified at creation time cannot be changed later.

#### [UltraLite Connection Parameters \[page 181\]](#)

UltraLite supports these connection parameters when connecting to an UltraLite database.

#### [UltraLite Database Properties \[page 203\]](#)

UltraLite database property values are defined when the database is first created.

#### [UltraLite Database Options \[page 206\]](#)

UltraLite database option values are defined when the database is first created and can be altered while connected to the database.

#### [UltraLite Utilities \[page 212\]](#)

UltraLite provides utilities that are designed to perform basic database administration activities at a command prompt. Many of these utilities share a similar functionality to the SQL Anywhere Server utilities. However, the way options are used can vary.

#### [UltraLite System Tables \[page 248\]](#)

The schema of an UltraLite database is stored in a proprietary format.

### 1.12.1 UltraLite Options

There are several options you can control when creating your UltraLite database. These are designed to help with the wide variety of UltraLite uses. Most options specified at creation time cannot be changed later.

You can specify creation options when creating a database using the `ulinit` or `ulload` utility, and from the supported client interfaces.

Boolean creation options are turned on with YES, Y, ON, TRUE, or 1, and are turned off with any of NO, N, OFF, FALSE, and 0. The options are case insensitive.

UltraLite creation options are specified as a semicolon separated list when creating a database from a programming interface.

You can use a prefix with an option to specify that the option applies only when an application is running on a particular type of platform.

Use the prefix `desktop:` with an option to indicate that the option only applies when the application is running on the desktop.

Use the prefix `device:` with an option to indicate that the option only applies when the application is running on the mobile device.

### **i** Note

If the option is appropriate for both desktop and mobile device, then do not use the prefix.

```
device:DBF=\Documents\sample.udb;desktop:DBF=c:\Databases
\sample.udb;UID=DBA;device:DBKEY=secret
```

Name	Description	Syntax
case	Sets the case-sensitivity of string comparisons in the UltraLite database.	<code>case=value</code>
checksum_level	Sets the level of checksum validation in the database. By default, checksum validation is enabled.	<code>checksum_level=value</code>
collation	Sets the collation sequence used by the UltraLite database. Setting this property with or without the UTF-8 property determines the character set of the database.	<code>collation=value</code>
date_format	Sets the default string format in which dates are retrieved from the database.	<code>date_format=value</code>
date_order	Controls the interpretation of date ordering of months, days, and years.	<code>date_order=value</code>
dbf	Specifies the path and file name for an UltraLite database.	<code>dbf=database-file</code>
dbkey	Provides an encryption key for the database.	<code>dbkey=string</code>
fips	Controls the use of FIPS-certified AES encryption.	<code>fips=value</code>
kdf_iterations	Makes it more difficult to access an encrypted database by prolonging each attack attempt.	<code>kdf_iterations=value</code>
max_hash_size	Sets the default index hash size in bytes.	<code>max_hash_size=value</code>



Name	Description	Syntax
mirror_file	Specifies the name of the database mirror file to which all database writes will be issued (at the same time as they are to the main database file).	<code>mirror_file=mirror-file</code>
nearest_century	Controls the interpretation of two-digit years in string-to-date conversions.	<code>nearest_century=value</code>
obfuscate	Controls whether data in the database is obfuscated. Obfuscation is not secure against skilled and determined attempts to gain access to the data.	<code>obfuscate=value</code>
page_size	Defines the database page size.	<code>page_size=sizek</code>
precision	Specifies the maximum number of digits in decimal point arithmetic results.	<code>precision=value</code>
pwd	Sets the password for the user.	<code>pwd=password</code>
reserve_size	Pre-allocates the file system space required for your UltraLite database, without actually inserting any data.	<code>reserve_size=number{ k   m   g }</code>
scale	Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.	<code>scale=value</code>
time_format	Sets the format for times retrieved from the database.	<code>time_format=value</code>
timestamp_format	Sets the format for timestamps retrieved from the database.	<code>timestamp_format=value</code>
timestamp_increment	Determines how the timestamp is truncated in UltraLite.	<code>timestamp_increment=value</code>
timestamp_with_time_zone_format	This option sets the format for TIME- STAMP WITH TIME ZONE values retrieved from the database.	<code>timestamp_with_time_zone_format=value</code>
uid	Sets the default user ID for the database.	<code>uid=user</code>
utf8_encoding	Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode.	<code>utf8_encoding=value</code>

#### In this section:

[UltraLite case Creation Option \[page 147\]](#)

Sets the case sensitivity of string comparisons in the UltraLite database.

[UltraLite checksum\\_level Creation Option \[page 149\]](#)

Specify the level of checksum validation for the database.

[UltraLite collation Creation Option \[page 150\]](#)

Sets the database collation.

[UltraLite date\\_format Creation Option \[page 151\]](#)

Specify the format used for converting date values to strings.

[UltraLite date\\_order Creation Option \[page 153\]](#)

Specify the default order of date parts when interpreting a date string.

[UltraLite DBF Creation Option \[page 155\]](#)

Specify the path and file name for an UltraLite database.

[UltraLite DBKEY Creation Option \[page 156\]](#)

When creating a new UltraLite database, this creation option provides an encryption key for the database.

[UltraLite Desktop Creation Option Prefix \[page 157\]](#)

Use the prefix desktop: with an UltraLite option to indicate that the option only applies when the application is running on the desktop.

[UltraLite Device Creation Option Prefix \[page 158\]](#)

Use the prefix device: with an UltraLite option to indicate that the option only applies when the application is running on the mobile device.

[UltraLite fips Creation Option \[page 159\]](#)

Controls whether the new database should be encrypted using AES or AES\_FIPS strong encryption.

[UltraLite kdf\\_iterations Creation Option \[page 160\]](#)

Specify the number of iterations, in thousands, for the key derivation function that converts the pass phrase provided by the DBKEY option into an actual encryption key.

[UltraLite max\\_hash\\_size Creation Option \[page 161\]](#)

Specify the maximum default primary key or index hash size in bytes.

[UltraLite nearest\\_century Creation Option \[page 162\]](#)

Specify the interpretation of two-digit years in string-to-date conversions.

[UltraLite PWD Creation Option \[page 164\]](#)

Specify the password for the default user.

[UltraLite obfuscate Creation Option \[page 165\]](#)

Specify simple obfuscation for the data in the database.

[UltraLite page\\_size Creation Option \[page 166\]](#)

Specify the database page size in kilobytes.

[UltraLite Precision Creation Option \[page 168\]](#)

Specify the maximum number of digits in decimal point arithmetic results.

[UltraLite scale Creation Option \[page 169\]](#)

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

[UltraLite time\\_format Creation Option \[page 171\]](#)

Specify the format used for converting time values to strings.

[UltraLite timestamp\\_format Creation Option \[page 173\]](#)

Specify the format used for converting timestamp values to strings.

[UltraLite timestamp\\_increment Creation Option \[page 175\]](#)

Specify the limit on the resolution of timestamp values. As timestamps are inserted into the database, UltraLite truncates them to match this increment.

[UltraLite timestamp\\_with\\_time\\_zone\\_format Creation Option \[page 177\]](#)

Specify the format used for converting `TIMESTAMP WITH TIME ZONE` values to strings.

[UltraLite UID Creation Option \[page 179\]](#)

Specify the default user ID for the database.

[UltraLite utf8\\_encoding Creation Option \[page 180\]](#)

Specify UTF-8 encoding (8-bit multibyte encoding for Unicode) for the database.

## Related Information

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite Character Sets \[page 32\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

[UltraLite DBF Connection Parameter \[page 191\]](#)

[UltraLite DBKEY Connection Parameter \[page 192\]](#)

[UltraLite Desktop Connection Parameter Prefix \[page 194\]](#)

[UltraLite Device Connection Parameter Prefix \[page 195\]](#)

[UltraLite MIRROR\\_FILE Connection Parameter \[page 196\]](#)

[UltraLite PWD Connection Parameter \[page 198\]](#)

[UltraLite RESERVE\\_SIZE Connection Parameter \[page 199\]](#)

[UltraLite UID Connection Parameter \[page 202\]](#)

[Accessing Database Options \[page 45\]](#)

[Reading Database Properties \[page 43\]](#)

### 1.12.1.1 UltraLite case Creation Option

Sets the case sensitivity of string comparisons in the UltraLite database.

## Syntax

```
case=value
```

## Allowed Values

Ignore, Respect

## Default

Ignore

## Remarks

The case sensitivity of data is reflected in tables, indexes, and so on. By default, UltraLite databases perform case-insensitive comparisons, although data is always held in the case in which you enter it. Identifiers (such as table and column names) and user IDs are always case insensitive, regardless of the database case sensitivity. Passwords are always case sensitive, regardless of the case sensitivity of the database.

The results of comparisons on strings, and the sort order of strings, depend in part on the case sensitivity of the database.

There are some collations where particular care is required when assuming case insensitivity of identifiers. In particular, Turkish collations have a case-conversion behavior that can cause unexpected and subtle errors. The most common error is that a system object containing a letter *i* or *I* is not found.

You cannot change the case of an existing database. Instead, you must create a new database.

From SQL Central, you can set the case sensitivity in any wizard that creates a database. On the [New Database Collation And Character Set](#) page, click the [Use Case-sensitive String Comparisons](#) option.

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class. Pass in [case=respect](#) to the creation string parameter of the CreateDatabase method in your programming interface (or [case=ignore](#) for a case-insensitive database).

## Related Information

[Strings in UltraLite \[page 257\]](#)

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(uload\) \[page 234\]](#)

## 1.12.1.2 UltraLite checksum\_level Creation Option

Specify the level of checksum validation for the database.

### Syntax

```
checksum_level=value
```

### Allowed Values

0, 1, 2

### Default

2

### Remarks

Checksums are used to detect offline corruption on pages stored to disk, flash, or memory, which can help reduce the chances of other data being corrupted as the result of a bad critical page. Depending on the level you choose, UltraLite calculates and records a checksum for each database page before it writes the page to storage.

If the calculated checksum does not match the stored checksum for a page read from storage, then the page has been modified or became corrupted during the storage/retrieval of the page. If a checksum validation fails, then when the database loads a page, UltraLite stops the database and reports a fatal error. This error cannot be corrected; you must re-create your UltraLite database and report the database failure to SAP.

If you unload and reload an UltraLite database with checksums enabled, the checksum level is preserved and restored.

The following values are supported for the checksum\_level:

#### 0

Do not add checksums to database pages.

#### 1

Add checksums to important database pages, such as indexes and synchronization status pages, but not row pages.

#### 2

Add checksums to all database pages (the default).

From SQL Central, you can configure the use of checksums in any wizard that creates a database. On the [New database storage settings](#) page of the *Create Database Wizard*, click the *Checksum level for database pages* option.

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class.

## Related Information

[UltraLite Performance Tips \[page 569\]](#)

[UltraLite Database Connections \[page 39\]](#)

[UltraLite Database Properties \[page 203\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(uload\) \[page 234\]](#)

[UltraLite page\\_size Creation Option \[page 166\]](#)

### 1.12.1.3 UltraLite collation Creation Option

Sets the database collation.

## Syntax

```
collation=value
```

## Allowed Values

String

## Default

1252Latin1

## Remarks

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class.

You can view a list of collations supported by UltraLite using the following command:

```
ulinit -Z
```

You can set the collation using SQL Central. Go to the [New Database Collation And Character Set](#) page, click either the default collation (1252Latin1), or select an alternate one from the list.

## Related Information

[UltraLite Character Sets \[page 32\]](#)

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite Supported Collations \[page 34\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

[UltraLite utf8\\_encoding Creation Option \[page 180\]](#)

### 1.12.1.4 UltraLite date\_format Creation Option

Specify the format used for converting date values to strings.

## Syntax

```
date_format=value
```

## Allowed Values

String

## Default

YYYY-MM-DD (this corresponds to ISO date format specifications)

## Remarks

The default date format YYYY-MM-DD conforms to ISO 8601. For example, "January 7, 2006" in this format is presented as "2006-01-07". You can specify a different format and order for year, month, and day.

The format is a string using the following symbols:

Symbol	Description
YY	Two digit year.
YYYY	Four digit year.
MM	Two digit month, or two digit minutes if following a colon (as in hh:mm).
MMM[m...]	Character short form for month. As many characters as there are "m"s. An uppercase M causes the output to be made uppercase.
D	Single digit day of week, (0 = Sunday, 6 = Saturday).
DD	Two digit day of month. A leading zero is not required.
DDD[d...]	Character short form for day of the week. An uppercase D causes the output to be made uppercase.
JJJ	Julian day of the year, from 1 to 366.

You cannot change the date format of an existing database. Instead, you must create a new database.

Allowed values are constructed from the symbols listed in the table above. Each symbol is substituted with the appropriate data for the date that is being formatted.

### Controlling output case

For symbols that represent character data (such as MMM), you can control the case of the output as follows:

- Type the symbol in uppercase to have the format appear in uppercase. For example, MMM produces JAN.
- Type the symbol in lowercase to have the format appear in lowercase. For example, mmm produces jan.
- Type the symbol in mixed case to have UltraLite choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.

### Controlling zero-padding

For symbols that represent numeric data, you can control zero-padding with the case of the symbols:

- Type the symbol in same-case (such as MM or mm) to allow zero padding. For example, yyyy/mm/dd could produce 2002/01/01.
- Type the symbol in mixed case (such as Mm) to suppress zero padding. For example, yyyy/Mm/Dd could produce 2002/1/1.

From SQL Central, you can set the date format in any wizard that creates a database. On the [New database creation parameters](#) page, click the *Date Format* option.

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class.



## Example

The following table illustrates `date_format` settings, together with the output from a `SELECT CURRENT DATE` statement, executed on Thursday May 21, 2001.

<code>date_format</code> syntax used	Result returned
<code>YYYY/MM/DD/ddd</code>	2001/05/21/thu
<code>JJJ</code>	141
<code>mmm YYYY</code>	may 2001
<code>MM-YYYY</code>	05-2001

## Related Information

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite `date\_order` Creation Option \[page 153\]](#)

[UltraLite Initialize Database Utility \(`ulinit`\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(`ulload`\) \[page 234\]](#)

### 1.12.1.5 UltraLite `date_order` Creation Option

Specify the default order of date parts when interpreting a date string.

#### Syntax

```
date_order=value
```

#### Allowed Values

MDY, YMD, DMY

#### Default

YMD (this corresponds to ISO date format specifications)

## Remarks

The default order for year, month, and day corresponds to the ISO 8601 date format. For example, "06-01-07" is interpreted as January 7, 2006.

You can specify a different order for the interpretation of date parts. For example, if "06-01-07" represents June 1, 2007 then specify "MDY" for the date order.

You can only specify the date order for a new database. Once the database has been created, you cannot change the date order.

From SQL Central, you can set the date order in any wizard that creates a database. On the [New database creation parameters](#) page, click the [Date Order](#) option.

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class.

## Example

Different values determine how the date 10/11/12 is interpreted:

Date order	Interpretation
MDY	October 11 2012
YMD	November 12 2010
DMY	November 10 2012

Use the nearest\_century option to control the interpretation of two-digit years in string-to-date conversions.

The following SQL query returns 2010-11-12 using the default date\_order and nearest\_century settings.

```
SELECT CAST( CAST( '10/11/12' AS DATE ) AS VARCHAR(15) );
```

## Related Information

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite date\\_format Creation Option \[page 151\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

## 1.12.1.6 UltraLite DBF Creation Option

Specify the path and file name for an UltraLite database.

Use this creation option to specify the path and file name for a new database file or when connecting to an existing database file.

### Syntax

```
DBF=u.l-db
```

### Behavior

1. On connect, look to see if the database is already running. If DBN is specified, look for a matching database and connect if found, proceed to auto-start if not.
2. If DBF is specified, look for a matching database (identical filename) and connect if found, proceed to auto-start if not.
3. If neither DBN nor DBF is specified, and a single database is running, connect to it.
4. A database is auto-started when required if DBF is specified. If DBN is also specified, it becomes the name of the running database, otherwise a name is generated from the base filename.

### Remarks

If you are connecting to multiple databases on different devices from a single connection string, you can use the following options to name platform-specific alternates:

- desktop:DBF
- device:DBF

If specified, these platform-specific creation options take precedence over DBF.

The value of DBF must meet the file name requirements for the platform.

#### Microsoft Windows Mobile

If you are deploying to a Microsoft Windows Mobile device, UltraLite administration tools running on the Microsoft Windows desktop can connect to an UltraLite database on an attached Microsoft Windows Mobile device. To identify a file on a Microsoft Windows Mobile device, you must specify the required absolute path, and use the [wce:](#) file prefix.

You cannot use the [wce:](#) file prefix in an application running on the Microsoft Windows Mobile device.

Any leading or trailing spaces in option values are ignored. The value cannot include leading single quotes, leading double quotes, or semicolons.

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite Options \[page 143\]](#)

### 1.12.1.7 UltraLite DBKEY Creation Option

When creating a new UltraLite database, this creation option provides an encryption key for the database.

#### Syntax

```
DBKEY=string
```

#### Default

No key is provided.

#### Remarks

If a database is created using an encryption key, the database file is strongly encrypted by using either the 256-bit AES or FIPS-certified 256-bit AES algorithm. By using strong encryption, you have increased security against skilled and determined attempts to gain access to the data.

Any leading or trailing spaces in option values are ignored. The value cannot include leading single quotes, leading double quotes, or semicolons.

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[Database Security \[page 35\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite Options \[page 143\]](#)

[UltraLite obfuscate Creation Option \[page 165\]](#)

## 1.12.1.8 UltraLite Desktop Creation Option Prefix

Use the prefix `desktop:` with an UltraLite option to indicate that the option only applies when the application is running on the desktop.

### Syntax

```
desktop:option=value
```

### Remarks

If the option is appropriate for both desktop and mobile device, then do not use the prefix.

Use the `desktop` option prefix for UltraLite client applications that run on a variety of devices.

Options with a `desktop` or `device` prefix take precedence over options without a prefix.

### Example

The following example identifies different database files for the desktop and the mobile device, the location of the temporary directory on the desktop, and the `cache_size` for the mobile device:

```
"desktop:DBF=C:\dir\db.udb;device:DBF=\SD Card\db.udb;desktop:temp_dir=\Temp;device:cache_size=4M"
```

### Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite File Path Formats in Connection Parameters \[page 42\]](#)

[Precedence of Connection Parameters for UltraLite Administration Tools \[page 41\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite Options \[page 143\]](#)

## 1.12.1.9 UltraLite Device Creation Option Prefix

Use the prefix `device:` with an UltraLite option to indicate that the option only applies when the application is running on the mobile device.

### Syntax

```
device:option=value
```

### Remarks

If the option is appropriate for both desktop and mobile device, then do not use the prefix.

Use the `device` option prefix for UltraLite client applications that run on a variety of devices.

Options with a `desktop` or `device` prefix take precedence over options without a prefix.

### Example

The following example identifies different database files for the desktop and the mobile device, the location of the temporary directory on the mobile device, and the `cache_size` for the mobile device:

```
"desktop:DBF=C:\dir\db.udb;device:DBF=\SD Card\db.udb;device:temp_dir=\Temp;device:cache_size=4M"
```

### Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite File Path Formats in Connection Parameters \[page 42\]](#)

[Precedence of Connection Parameters for UltraLite Administration Tools \[page 41\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite Options \[page 143\]](#)

[UltraLite DBF Connection Parameter \[page 191\]](#)

## 1.12.1.10 UltraLite fips Creation Option

Controls whether the new database should be encrypted using AES or AES\_FIPS strong encryption.

### Syntax

```
fips=value
```

### Allowed Values

Yes (use AES\_FIPS), No (use AES)

### Default

No

### Remarks

This option is not supported by UltraLiteJ, or UltraLite for Apple iOS.

The only way to change the type of database encryption is to recreate the database with the appropriate fips or obfuscate creation option. You can change the database encryption key by specifying a new encryption key on the Connection object. Users connecting to the database must supply the key each time they connect.

From SQL Central, you can configure encryption in any wizard that creates a database. On the [New database storage settings](#) page of the [Create Database Wizard](#), select the [Encrypt the database](#) and [Use strong encryption](#) options and then select the type of AES encryption. You must also specify and confirm the encryption key.

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class.

To deploy a database with the fips option enabled, copy all appropriate libraries for your platform.

### Related Information

[Simple Obfuscation Versus Strong Encryption](#)

[Database Security \[page 35\]](#)

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite obfuscate Creation Option \[page 165\]](#)

[UltraLite DBKEY Connection Parameter \[page 192\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(uload\) \[page 234\]](#)

## 1.12.1.11 UltraLite kdf\_iterations Creation Option

Specify the number of iterations, in thousands, for the key derivation function that converts the pass phrase provided by the DBKEY option into an actual encryption key.

### Syntax

```
kdf_iterations=value
```

### Allowed Values

1 to 1000

### Default

The default value for Apple macOS and iOS is 30, which results in 30000 iterations.

The default value for other platforms, including desktop and device platforms, is 5, which results in 5000 iterations.

### Remarks

This parameter is specified only at database creation.

The key derivation function makes it more difficult to access an encrypted database by prolonging each attack attempt.

#### **i** Note

A larger number of iterations will make passwords harder to break through brute force, but will increase database start-up time.



There are two cases when an explicit setting may be required:

1. You are using a very slow device and UltraLite takes too long to start with encryption. For example:

```
kdf_iterations=1
```

2. You are using a high-end computer and want added security. For example, when running UltraLite on a Windows or Linux desktop:

```
kdf_iterations=100
```

### 1.12.1.12 UltraLite max\_hash\_size Creation Option

Specify the maximum default primary key or index hash size in bytes.

#### Syntax

```
max_hash_size=value
```

#### Allowed Values

0 to 32

#### Default

4

#### Remarks

A hash is an optional part of an index entry that is stored in the index page. The hash transforms the actual row values for the indexed columns into a numerical equivalent (a key), while still preserving ordering for that index. The size of the key, and how much of the actual value UltraLite hashes, is determined by the hash size you set.

A row ID allows UltraLite to locate the row for the actual data in the table. A row ID is always part of an index entry. If you set the hash size to 0 (disable index hashing), then the index entry only contains this row ID. For all other hash sizes, the hash key, which can contain all or part of the transformed data in that row, is stored along with the row ID in the index page. You can improve query performance on these indexed columns because UltraLite may not always need to find, load, and unpack data before it can compare actual row values.

Determining an appropriate default database hash size requires that you evaluate the trade-off between query efficiency and database size: the higher the maximum hash value, the larger the database size grows.

UltraLite only uses as many bytes as required for the data type(s) of the column(s), up to the maximum value specified by this option. The default hash size is only used if you do not set a size when you create the index. If you set the default hash size to 0, UltraLite does not hash row values.

You cannot change the hash size for an existing index. When creating a primary key or new index, you can override the default value with the UltraLite *Set Primary Key Wizard* or *Create Index Wizard* in SQL Central, or with the WITH MAX SIZE clause of a CREATE INDEX or a CREATE TABLE statement.

If you declare your columns as DOUBLE, FLOAT, or REAL, no hashing is used. The hash size is always ignored.

From SQL Central, you can set the maximum hash size in any wizard that creates a database. On the *New database storage settings* page of the *Create Database Wizard*, click the *Default maximum hash size for indexes* option.

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class.

## Related Information

[UltraLite Performance Tips \[page 569\]](#)

[UltraLite Indexes \[page 62\]](#)

[Optimal Hash Size Limit \[page 573\]](#)

[How to Access Creation Option Values \[page 32\]](#)

[CREATE INDEX Statement \[UltraLite\] \[page 531\]](#)

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

### 1.12.1.13 UltraLite nearest\_century Creation Option

Specify the interpretation of two-digit years in string-to-date conversions.

## Syntax

```
nearest_century=value
```

## Allowed Values

Integer, between 0 and 100, inclusive

## Default

50

## Remarks

UltraLite automatically converts a string into a date when a date value is expected, even if the year is represented in the string by only two digits. For a two-digit date, you need to set the appropriate rollover value. Two digit years less than the value are converted to 20`yy`, while years greater than or equal to the value are converted to 19`yy`.

Choosing an appropriate rollover value typically is determined by:

### The use of two-digit dates

Otherwise, nearest century conversion isn't applicable. Two-digit years less than the `nearest_century` value you set are converted to 20`yy`, while years greater than or equal to the value are converted to 19`yy`.

Store four-digit dates to avoid issues with incorrect conversions.

### Consolidated database compatibility

For example, the historical SQL Anywhere behavior is to add 1900 to the year. Adaptive Server Enterprise behavior is to use the nearest century, so for any year where value `yy` is less than 50, the year is set to 20`yy`.

### What the date represents: past event or future event

Birth years are typically those that would require a lower rollover value since they occur in the past. So for any year where `yy` is less than 20, the year should be set to 20`yy`. However, if the date is used as an expiry date, then having a higher value would be a logical choice, since the date is occurring in the future.

You cannot change the nearest century of an existing database. Instead, you must create a new database.

From SQL Central, you can configure the nearest century setting in any wizard that creates a database. On the [New database creation parameters](#) page, click the *Nearest Century* option.

From a client application, set this option as one of the creation options for the `CreateDatabase` method on the `DatabaseManager/ULDatabaseManager` class.

## Related Information

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(uload\) \[page 234\]](#)

## 1.12.1.14 UltraLite PWD Creation Option

Specify the password for the default user.

### Syntax

```
PWD=password
```

### Default

If you do not set both the user ID and password, UltraLite opens connections with *UID=DBA* and *PWD=sql*.

### Remarks

For Android devices, you can use `Configuration.setPassword` as an alternative to setting this creation option.

Every user of a database has a password. UltraLite supports up to four user ID/password combinations.

You can set passwords to NULL or an empty string.

A random 4-byte salt value is generated when a new user is created or an existing user changes their password. The salt value is appended to the user's password when calculating the password hash and is stored in the database along with the hash. Salting significantly decreases vulnerability to dictionary attacks and also ensures that users with the same password will have different password hashes.

This creation option is not encrypted. However, UltraLite hashes the password before saving it, so you can only modify a password using SQL Central.

### Related Information

[Users](#)

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Users \[page 66\]](#)

[User Authentication \[page 697\]](#)

[UltraLite Database Connections \[page 647\]](#)

[User Authentication \[page 697\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite UID Connection Parameter \[page 202\]](#)

[UltraLite UID Connection Parameter \[page 202\]](#)

## 1.12.1.15 UltraLite obfuscate Creation Option

Specify simple obfuscation for the data in the database.

### Syntax

```
obfuscate=value
```

### Allowed Values

Boolean.

### Default

0 (databases are not obfuscated)

### Remarks

Obfuscation makes it difficult for someone using a disk utility to look at the file to decipher the data in your database. However, obfuscation is not secure against skilled and determined attempts to gain access to the data. Simple obfuscation does not require a key to encode the database.

You must use strong encryption to make the database inaccessible without the correct encryption key.

From SQL Central, you can configure obfuscation in any wizard that creates a database. On the [New database storage settings](#) page of the [Create Database Wizard](#), select the [Encrypt the database](#) and [Use simple encryption \(obfuscation\)](#) options.

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class.

## Related Information

[Database Security \[page 35\]](#)

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite fips Creation Option \[page 159\]](#)

[UltraLite DBKEY Connection Parameter \[page 192\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(uload\) \[page 234\]](#)

### 1.12.1.16 UltraLite page\_size Creation Option

Specify the database page size in kilobytes.

#### Syntax

```
page_size=sizeK
```

#### Allowed Values

1K, 2K, 4K, 8K, 16K

#### Default

4K

#### Remarks

The page size must be entered with a K or k after the digit, or, alternatively, the equivalent number of bytes (1024, 2048, 4096, 8192, or 16384).

UltraLite databases are stored in pages, and all I/O operations are carried out a page at a time. The page size you choose can affect the performance or size of the database.

If you use any value other than those listed, the size is changed to the next larger size. If you do not specify a unit, bytes are assumed.

If your platform has limited dynamic memory, consider using a smaller page size to limit the effect on synchronization memory requirements.

When choosing a page size, you should keep the following guidelines in mind:

#### **Database size**

Larger databases usually benefit from a larger page size. Larger pages hold more information and therefore use space more effectively, particularly if you insert rows that are slightly more than half a page in size. The larger the page, the less page swapping that is required.

#### **Number of rows**

Because a row (excluding BLOBs) must fit on a page, the page size determines how large the largest packed row can be, and how many rows you can store on each page. Sometimes reading one page to obtain the values of one row may have the side effect of loading the contents of the next few rows into memory.

#### **Query types**

In general, smaller page sizes are likely to benefit queries that retrieve a relatively small number of rows from random locations. By contrast, larger pages tend to benefit queries that perform sequential table scans.

#### **Cache size**

Large page sizes may require larger cache sizes. With dynamic cache sizing, UltraLite grows the cache as required.

#### **Index entries**

Page size also affects indexes. The larger the database page, the more index entries it can hold.

#### **Device memory**

Small pages are particularly useful if your database must run on small devices with limited memory. For example, 1 MB of memory can hold 1000 pages that are each 1 KB in size, but only 250 pages that are 4 KB in size.

You cannot change the page size of an existing database. Instead, you must create a new database.

From SQL Central, you can set the page size in any wizard that creates a database. On the [New database storage settings](#) page of the [Create Database Wizard](#), click the desired page size.

From a client application, set this option as one of the creation options for the `CreateDatabase` method on the `DatabaseManager/ULDatabaseManager` class.

## **Example**

To set the page size of the database to 8 KB, specify `page_size=8k` or `page_size=8192`:

```
ulinit test.udb --page_size=8k
```

## **Related Information**

[Row Packing and Table Definitions \[page 54\]](#)

- [UltraLite Indexes \[page 62\]](#)
- [How to Access Creation Option Values \[page 32\]](#)
- [UltraLite case Creation Option \[page 147\]](#)
- [UltraLite CACHE\\_SIZE Connection Parameter \[page 187\]](#)
- [UltraLite CACHE\\_MIN\\_SIZE Connection Parameter \[page 186\]](#)
- [UltraLite CACHE\\_MAX\\_SIZE Connection Parameter \[page 185\]](#)
- [UltraLite RESERVE\\_SIZE Connection Parameter \[page 199\]](#)
- [UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)
- [UltraLite Load XML to Database Utility \(uload\) \[page 234\]](#)

## 1.12.1.17 UltraLite Precision Creation Option

Specify the maximum number of digits in decimal point arithmetic results.

### Syntax

```
precision=value
```

### Allowed Values

Integer, between 1 and 127, inclusive

### Default

30

### Remarks

The position of the decimal point is determined by the precision and the scale of the number: precision is the total number of digits to the left and right of the decimal point; scale is the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

Choosing an appropriate decimal point position is typically determined by:

#### **The type of arithmetic procedures you perform**

Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum precision.



For example, when a DECIMAL(8,2) is multiplied with a DECIMAL(9,2), the result could require a DECIMAL(17,4). If precision is 15, only 15 digits are kept in the result. If scale is 4, the result is a DECIMAL(15,4). If scale is 2, the result is a DECIMAL(15,2). In both cases, there is a possibility of an overflow error.

#### The relationship between scale and precision values

The scale sets the number of digits in the fractional part of the number, and cannot be negative or greater than the precision.

You cannot change the precision of an existing database. Instead, you must create a new database.

If you are using an Oracle database as the consolidated database, all UltraLite remotes and the Oracle consolidated database must have the same precision value.

From SQL Central, you can set the precision in any wizard that creates a database. On the [New database creation parameters](#) page, click the [Precision](#) option.

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class.

## Related Information

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

### 1.12.1.18 UltraLite scale Creation Option

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

#### Syntax

```
scale=value
```

#### Allowed Values

Integer, between 0 and 127, inclusive

## Default

6

## Remarks

The position of the decimal point is determined by the precision and the scale of the number: precision is the total number of digits to the left and right of the decimal point; scale is the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

Choosing an appropriate decimal point position is typically determined by:

### **The type of arithmetic procedures you perform**

Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum precision.

For example, when a DECIMAL(8,2) is multiplied with a DECIMAL(9,2), the result could require a DECIMAL(17,4). If precision is 15, only 15 digits are kept in the result. If scale is 4, the result is a DECIMAL(15,4). If scale is 2, the result is a DECIMAL(15,2). In both cases, there is a possibility of an overflow error.

### **The relationship between scale and precision values**

The scale sets the number of digits in the fractional part of the number, and cannot be negative or greater than the precision.

You cannot change the scale of an existing database. Instead, you must create a new database.

From SQL Central, you can set the scale in any wizard that creates a database. On the [New database creation parameters](#) page, click the *Scale* option.

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class.

## Example

When a DECIMAL(8,2) is multiplied with a DECIMAL(9,2), the result could require a DECIMAL(17,4). If precision is 15, only 15 digits are kept in the result. If scale is 4, the result is DECIMAL(15,4). If scale is 2, the result is a DECIMAL(15,2). In both cases, there is a possibility of overflow.

## Related Information

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite Precision Creation Option \[page 168\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

## 1.12.1.19 UltraLite time\_format Creation Option

Specify the format used for converting time values to strings.

### Syntax

```
time_format=value
```

### Allowed Values

String (composed of the symbols listed below)

### Default

HH:NN:SS.SSS

### Remarks

UltraLite writes times from time parts you set with the time\_format creation option. Time parts can include hours, minutes, seconds, and milliseconds.

TIME values can also be represented by strings. Before a time value can be retrieved, it must be assigned to a string variable.

UltraLite uses ISO 8601 as the default time standard. This international time standard indicates hours using the 24-hour clock system. For example, "midnight" in this international standard is written: 00:00:00. If you do not want to use the default time standard, you must specify a different format and order for these time parts.

The format is a string using the following symbols:

Symbol	Description
HH	Two digit hours (24 hour clock).
NN	Two digit minutes.
MM	Two digit minutes if following a colon (as in HH:MM).

Symbol	Description
SS[.sssss]	Seconds and fractions of a second, up to six decimal places. Not all platforms support timestamps to a precision of six places.

You cannot change the `time_format` creation option of an existing database. Instead, you must create a new database.

Each symbol is substituted with the appropriate data for the time that is being formatted. Any format symbol that represents character rather than digit output can be put in uppercase, which causes the substituted characters to be in uppercase. For numbers, using mixed case in the format string suppresses leading zeros.

Control zero-padding with the case of the symbols:

- Type the symbol in same-case (such as HH or hh) to allow zero padding. For example, HH:NN:SS could produce 01:01:01.
- Type the symbol in mixed case (such as Hh or hH) to suppress zero padding. For example, Hh:Nn:Ss could produce 1:1:1.
- Type the symbol in mixed case to have UltraLite choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.
- If the first two digits of the fractional seconds are mixed case (such as Ss or sSsss), then trailing zeros are removed. For example, hh:nn:ss.Sss could produce 12:34:56.1.

From SQL Central, you can set the `time_format` option in any wizard that creates a database. On the [New database creation parameters](#) page, click the *Time Format* option.

From a client application, set this option as one of the creation options for the `CreateDatabase` method on the `DatabaseManager/ULDatabaseManager` class.

## Example

The following command creates a database and sets the `time_format` creation option so that fractions of a second are excluded when retrieving TIME values from the database:

```
ulinit --time_format=HH:NN:SS example.udb
```

Execute the following query on the created database:

```
SELECT CAST(CAST('3:30:12.345 PM' AS TIME) AS CHAR(32))
```

The query returns 15:30:12.

## Related Information

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite timestamp\\_format Creation Option \[page 173\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

## 1.12.1.20 UltraLite timestamp\_format Creation Option

Specify the format used for converting timestamp values to strings.

### Syntax

```
timestamp_format=value
```

### Allowed Values

String

### Default

YYYY-MM-DD HH:NN:SS.SSS

### Remarks

The default timestamp format YYYY-MM-DD HH:NN:SS.SSS conforms to ISO 8601. For example, "January 7, 2006 12:34 AM" in this format is presented as "2006-01-07 00:34:00.000". You can specify a different format and order for year, month, day, and time parts.

The format is a string composed of the following symbols:

Symbol	Description
YY	Two digit year.
YYYY	Four digit year.
MM	Two digit month, or two digit minutes if following a colon (as in HH:MM).
MMM[m...]	Character short form for months. As many characters as there are "m"s. An uppercase M causes the output to be made uppercase.
D	Single digit day of week, (0 = Sunday, 6 = Saturday).

Symbol	Description
DD	Two digit day of month. A leading zero is not required.
DDD[d...]	Character short form for day of the week. An uppercase D causes the output to be made uppercase.
HH	Two digit hours. A leading zero is not required.
NN	Two digit minutes. A leading zero is not required.
SS[.sssss]	Seconds and parts of a second.
AA	Use 12 hour clock. Indicate times before noon with AM.
PP	Use 12 hour clock. Indicate times after noon with PM.
JJJ	Day of the year, from 1 to 366.

You cannot change the `timestamp_format` creation option of an existing database. Instead, you must create a new database.

Allowed values are constructed from the symbols listed in the table above. Each symbol is substituted with the appropriate data for the date that is being formatted.

For the character short forms, the number of letters specified is counted. The A.M. or P.M. indicator (which could be localized) is also truncated, if necessary, to the number of bytes corresponding to the number of characters specified.

For symbols that represent character data (such as MMM), control the case of the output as follows:

- Type the symbol in all uppercase to have the format appear in all uppercase. For example, MMM produces JAN.
- Type the symbol in all lowercase to have the format appear in all lowercase. For example, mmm produces jan.
- Type the symbol in mixed case to have UltraLite choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.

For symbols that represent numeric data, control zero-padding with the case of the symbols:

- Type the symbol in same-case (such as MM or mm) to allow zero padding. For example, yyyy/mm/dd could produce 2002/01/01.
- Type the symbol in mixed case (such as Mm) to suppress zero padding. For example, yyyy/Mm/Dd could produce 2002/1/1.
- Type the symbol in mixed case to have UltraLite choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.
- If the first two digits of the fractional seconds are mixed case (such as Ss or sSsss), then trailing zeros are removed. For example, hh:nn:ss.Sss could produce 12:34:56.1.

From SQL Central, you can set the `timestamp_format` option in any wizard that creates a database. On the [New database creation parameters](#) page, click the *Timestamp Format* option.

From a client application, set this option as one of the creation options for the `CreateDatabase` method on the `DatabaseManager/ULDatabaseManager` class.

## Example

The following command creates a database and sets the `timestamp_format` creation option so that the year is displayed in two digits and fractions of a second are excluded when retrieving `TIMESTAMP` values from the database:

```
ulinit --timestamp_format="YY-MM-DD HH:NN:SS" example.udb
```

Execute the following query on the created database:

```
SELECT CAST(CAST('Friday May 12, 2006 3:30 PM' AS TIMESTAMP) AS CHAR(32))
```

The query returns 06-05-12 15:30:00.

## Related Information

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite Concurrency \[page 583\]](#)

[Implementing Timestamp-based Downloads](#)

[UltraLite timestamp\\_increment Creation Option \[page 175\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(uload\) \[page 234\]](#)

### 1.12.1.21 UltraLite timestamp\_increment Creation Option

Specify the limit on the resolution of timestamp values. As timestamps are inserted into the database, UltraLite truncates them to match this increment.

## Syntax

```
timestamp_increment=value
```

## Allowed Values

1 to 60000000 microseconds

## Default

1 microsecond

## Remarks

Note that 1000000 microseconds equals 1 second.

You cannot change the `timestamp_increment` creation option of an existing database. Instead, you must create a new database.

This increment is useful when a `DEFAULT TIMESTAMP` column is being used as a primary key or row identifier.

From SQL Central, you can set the timestamp increment in any wizard that creates a database. On the [New database creation parameters](#) page, click the *Timestamp Increment* option.

From a client application, set this option as one of the creation options for the `CreateDatabase` method on the `DatabaseManager/ULDatabaseManager` class.

## Example

To store a value such as '2000/12/05 10:50:53.700', set this creation option to 100000. This value truncates the timestamp after the first decimal place in the seconds component.

## Related Information

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite Concurrency \[page 583\]](#)

[Implementing Timestamp-based Downloads](#)

[UltraLite `timestamp\_format` Creation Option \[page 173\]](#)

[UltraLite Initialize Database Utility \(`ulinit`\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(`ulload`\) \[page 234\]](#)



## 1.12.1.22 UltraLite timestamp\_with\_time\_zone\_format Creation Option

Specify the format used for converting TIMESTAMP WITH TIME ZONE values to strings.

### Syntax

```
timestamp_with_time_zone_format=value
```

### Allowed Values

String (composed of the symbols listed below)

### Default

YYYY-MM-DD HH:NN:SS.SSS+HH:NN

### Remarks

The default format YYYY-MM-DD HH:NN:SS.SSS+HH:NN conforms to ISO 8601. You can specify a different format and order for year, month, day, time, and time zone parts.

The format is a string using the following symbols:

Symbol	Description
YY	Two digit year
YYYY	Four digit year
MM	Two digit month, or two digit minutes if following a colon (as in HH:MM).
MMM[m...]	Character short form for months (as many characters as there are "m"s).
DD	Two digit day of month.
DDD[d...]	Character short form for day of the week.
HH	Two digit hours.
NN	Two digit minutes.

Symbol	Description
SS[.sssss]	Seconds and fractions of a second, up to six decimal places. Not all platforms support timestamps to a precision of six places.
AA	A.M. or P.M. (12 hour clock). Omit AA and PP for 24 hour time.
PP	P.M. if needed (12 hour clock). Omit AA and PP for 24 hour time.
HH	Two digit hours (time zone offset).
NN	Two digit minutes (time zone offset).

Each symbol is substituted with the appropriate data for the date that is being formatted.

For symbols that represent character data (such as MMM), you can control the case of the output as follows:

- Type the symbol in all uppercase to have the format appear in all uppercase. For example, MMM produces JAN.
- Type the symbol in all lowercase to have the format appear in all lowercase. For example, mmm produces jan.
- Type the symbol in mixed case to have UltraLite choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.
- If the first two digits of the fractional seconds are mixed case (such as Ss or sSsss) then trailing zeros are removed. For example, hh:nn:ss.Sss could produce 12:34:56.1.

If the character data is multibyte, the length of each symbol reflects the number of characters, not the number of bytes. For example, the MMM symbol specifies a length of three characters for the month.

For symbols that represent numeric data, control zero-padding with the case of the symbols:

- Type the symbol in same-case (such as MM or mm) to allow zero padding. For example, yyyy/mm/dd could produce 2002/01/01.
- Type the symbol in mixed case (such as Mm) to suppress zero padding. For example, yyyy/Mm/Dd could produce 2002/1/1.
- If the first two digits of the fractional seconds are mixed case (such as Ss or sSsss), then trailing zeros are removed. For example, hh:nn:ss.Sss could produce 12:34:56.1.

### **i** Note

If you change the setting for `timestamp_with_time_zone_format` option in a way that re-orders the date format, be sure to change the `date_order` option to reflect the same change, and vice versa.

## Example

The following command creates a database and sets the `timestamp_with_time_zone_format` creation option so that the year is displayed in two digits and fractions of a second are excluded when retrieving `TIMESTAMP WITH TIME ZONE` values from the database:

```
ulinit --timestamp_with_time_zone_format="YY-MM-DD HH:NN:SS+HH:NN" example.udb
```

Execute the following query on the created database:

```
SELECT CAST(CAST('Friday May 12, 2006 3:30 PM -04:00' AS TIMESTAMP WITH TIME ZONE) AS CHAR(32))
```

The query returns 06-05-12 15:30:00-04:00.

## Related Information

[date\\_order Option](#)

### 1.12.1.23 UltraLite UID Creation Option

Specify the default user ID for the database.

## Syntax

```
UID=user
```

## Default

If you do not set the UID and PWD when connecting, UltraLite opens connections with *UID=DBA* and *PWD=sql*.

## Remarks

Every user of a database has a user ID. UltraLite supports up to four user ID/password combinations.

UltraLite user IDs are separate from MobiLink user names and from other SQL Anywhere user IDs. You cannot change a user ID once it is created. Instead, you must delete the user ID and then add a new one.

You cannot set the UID to NULL or an empty string. The maximum length for a user ID is 31 characters. User IDs are case insensitive.

Any leading or trailing spaces in option values are ignored. This creation option's value cannot include leading single quotes, leading double quotes, or semicolons.

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Users \[page 66\]](#)

[User Authentication \[page 697\]](#)

[UltraLite Database Connections \[page 647\]](#)

[User Authentication \[page 697\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite Options \[page 143\]](#)

### 1.12.1.24 UltraLite utf8\_encoding Creation Option

Specify UTF-8 encoding (8-bit multibyte encoding for Unicode) for the database.

#### Syntax

```
utf8_encoding=value
```

#### Values

Boolean.

#### Default

1 (databases are UTF-8 encoded)

#### Remarks

UTF-8 characters are represented by one to four bytes. For other multibyte collations, one or two bytes are used. For all provided multibyte collations, characters of two or more bytes are considered to be alphabetic. You can use these characters in identifiers without requiring double quotes.

Characters in an UltraLite database are either from the codepage implicit in the chosen collation, or are UTF8 encoded. UltraLite databases that use the UTF8BIN collation are automatically UTF8 encoded. If the operating system to which you are deploying your UltraLite application uses UTF8 or Unicode (like most Linux

distributions, Microsoft Windows Mobile, and Apple iOS) or if you plan to store characters from multiple languages in your database, you should create your database using a UTF8 encoding. If you try synchronizing UTF-8 encoded characters into a consolidated table that does not support Unicode, a user error is reported.

From SQL Central, you can choose UTF-8 encoding in any wizard that creates a database. On the [New database collation and character set](#) page, click the *Yes, use UTF-8 as the database character set* option.

From a client application, set this option as one of the creation options for the CreateDatabase method on the DatabaseManager/ULDatabaseManager class.

## Related Information

[UltraLite Platform Requirements for Character Set Encoding \[page 33\]](#)

[UltraLite Character Sets \[page 32\]](#)

[How to Access Creation Option Values \[page 32\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

## 1.12.2 UltraLite Connection Parameters

UltraLite supports these connection parameters when connecting to an UltraLite database.

You can use a prefix with a connection parameter to specify that the parameter applies only when an application is running on a particular type of platform.

Use the prefix `desktop:` with a connection parameter to indicate that the parameter only applies when the application is running on the desktop.

Use the prefix `device:` with a connection parameter to indicate that the parameter only applies when the application is running on the mobile device.

### **i** Note

If the parameter is appropriate for both desktop and mobile device, then do not use the prefix.

```
device:DBF=\Documents\sample.udb;desktop:DBF=c:\Databases
\sample.udb;UID=DBA;device:DBKEY=secret
```

Parameter name	Description
CACHE_MAX_SIZE	Specifies the maximum amount of memory to allocate for the file cache.
CACHE_MIN_SIZE	Specifies the minimum amount of memory to allocate for the file cache.

Parameter name	Description
CACHE_SIZE	Specifies the initial amount of memory to allocate for the file cache.
COMMIT_FLUSH	Defines which transactions are recovered following a hardware failure or crash.
CON	Names a connection so that switching to it is easier in multi-connection applications.
DBF	<p>At creation time, this parameter sets the location of the database.</p> <p>For subsequent connections, this parameter tells UltraLite where to find the database file.</p> <p>You can use DBF if you are creating a single-platform application or are connecting to a database from an UltraLite administration tool. Use the desktop: or device: prefixes if you are programming an UltraLite client that connects to different platform-specific databases.</p>
DBKEY	<p>At creation-time, this parameter sets the encryption key used to encrypt the database.</p> <p>For subsequent connections, it specifies the encryption key used to encrypt the database.</p>
DBN	Differentiates databases by name when applications connect to more than one database.
MIRROR_FILE	Specifies the name of the database mirror file to which all database writes will be issued (at the same time as they are to the main database file).
PWD	When creating a new UltraLite database, this connection parameter sets the password for the default user. When connecting to an existing database, it defines the password for a user ID that is used for authentication.
RESERVE_SIZE	Pre-allocates the file system space required for the UltraLite database without actually inserting any data.
START	Starts the UltraLite engine executable.
TEMP_DIR	Specifies the name of the directory (which must already exist) into which UltraLite will place a temporary file (with a name derived from the database name).

Parameter name	Description
UID	When creating a new UltraLite database, this connection parameter sets the default user ID for the database. When connecting to an existing database, it specifies the user ID with which you connect to the database.

#### In this section:

##### [UltraLite Connection Parameter Prefixes \[page 184\]](#)

You can use a prefix with a connection parameter to specify that a parameter applies only when an application is running on a particular type of platform.

##### [UltraLite CACHE\\_MAX\\_SIZE Connection Parameter \[page 185\]](#)

Defines the maximum size of the database cache. UltraLite manages the cache size automatically, so setting this parameter should not be necessary.

##### [UltraLite CACHE\\_MIN\\_SIZE Connection Parameter \[page 186\]](#)

Defines the minimum size of the database cache. UltraLite manages the cache size automatically, so setting this parameter should not be necessary.

##### [UltraLite CACHE\\_SIZE Connection Parameter \[page 187\]](#)

Defines the initial size of the database cache. UltraLite manages the cache size automatically, so setting this parameter should not be necessary.

##### [UltraLite COMMIT\\_FLUSH Connection Parameter \[page 188\]](#)

Determines when committed transactions are flushed to storage after a commit call. If no calls to commit are made by the UltraLite application, no flush can occur.

##### [UltraLite CON Connection Parameter \[page 190\]](#)

Names a connection so that switching to it is easier in multi-connection applications.

##### [UltraLite DBF Connection Parameter \[page 191\]](#)

Specify the path and file name for an UltraLite database.

##### [UltraLite DBKEY Connection Parameter \[page 192\]](#)

When creating a new UltraLite database, this connection parameter provides an encryption key for the database.

##### [UltraLite DBN Connection Parameter \[page 193\]](#)

Differentiates databases by name when applications connect to more than one database.

##### [UltraLite Desktop Connection Parameter Prefix \[page 194\]](#)

Use the prefix desktop: with a connection parameter to indicate that the parameter only applies when the application is running on the desktop.

##### [UltraLite Device Connection Parameter Prefix \[page 195\]](#)

Use the prefix device: with a connection parameter to indicate that the parameter only applies when the application is running on the mobile device.

##### [UltraLite MIRROR\\_FILE Connection Parameter \[page 196\]](#)

Specifies the name of the database mirror file to which all database writes are be issued (at the same time as they are to the main database file).

##### [UltraLite PWD Connection Parameter \[page 198\]](#)

When creating a new UltraLite database, this connection parameter sets the password for the default user.

#### [UltraLite RESERVE\\_SIZE Connection Parameter \[page 199\]](#)

Pre-allocates the file system space required for your UltraLite database, without actually inserting any data. Reserving the file system space prevents the space from being used up by other files.

#### [UltraLite START Connection Parameter \[page 200\]](#)

Starts the UltraLite engine executable. This parameter is not supported for UltraLite for Android. This parameter is only required if the engine is not in one of the expected locations.

#### [UltraLite TEMP\\_DIR Connection Parameter \[page 201\]](#)

Specifies the name of the directory (which must already exist) into which UltraLite will place the temporary file (with a name derived from the database name).

#### [UltraLite UID Connection Parameter \[page 202\]](#)

When creating a new UltraLite database, this connection parameter sets the default user ID for the database.

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

### 1.12.2.1 UltraLite Connection Parameter Prefixes

You can use a prefix with a connection parameter to specify that a parameter applies only when an application is running on a particular type of platform.

Use the prefix `desktop:` with a connection parameter to indicate that the parameter only applies when the application is running on the desktop.

Use the prefix `device:` with a connection parameter to indicate that the parameter only applies when the application is running on the mobile device.

#### **i** Note

If the connection parameter is appropriate for both desktop and mobile device, then do not use the prefix.

```
device:DBF=\Documents\sample.udb;desktop:DBF=c:\Databases
\sample.udb;UID=DBA;device:CACHE_SIZE=100k
```

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Connection Parameters \[page 181\]](#)



## 1.12.2.2 UltraLite CACHE\_MAX\_SIZE Connection Parameter

Defines the maximum size of the database cache. UltraLite manages the cache size automatically, so setting this parameter should not be necessary.

### ☰, Syntax

```
CACHE_MAX_SIZE=number{ k | m }
```

### Default

The default maximum cache size is 20 MB for devices and 50 MB for desktops.

### Remarks

The `cache_max_size` connection parameter specifies the maximum amount of memory to allocate for the file cache. By default, the size is in bytes. Use `k` or `m` to specify units of kilobytes or megabytes.

If you exceed the maximum cache size, your platform's upper cache size limit is used instead. UltraLite does not grow the cache size beyond the actual file size of the database.

If you specify a cache size limit that is greater than the size of your database, the excess space might be used for caching rows.

Any leading or trailing spaces in connection parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

### Example

The following connection string fragment sets the maximum cache size to 100 MB.

```
"CACHE_MAX_SIZE=100m"
```

### Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[Cache Size Adjustment for an UltraLite Database \[page 569\]](#)

[UltraLite CACHE\\_MIN\\_SIZE Connection Parameter \[page 186\]](#)

[UltraLite CACHE\\_SIZE Connection Parameter \[page 187\]](#)

[UltraLite cache\\_allocation Option \[page 207\]](#)

[UltraLite page\\_size Creation Option \[page 166\]](#)

[UltraLite RESERVE\\_SIZE Connection Parameter \[page 199\]](#)

### 1.12.2.3 UltraLite CACHE\_MIN\_SIZE Connection Parameter

Defines the minimum size of the database cache. UltraLite manages the cache size automatically, so setting this parameter should not be necessary.

#### ☞ Syntax

```
CACHE_MIN_SIZE=number{ k | m }
```

#### Default

The default cache size for devices is 256 KB. The default cache size for desktops is 512 KB.

#### Remarks

The `cache_min_size` connection parameter specifies the minimum amount of memory to allocate for the file cache. By default, the size is in bytes. Use `k` or `m` to specify units of kilobytes or megabytes.

If you set the minimum cache size to be greater than the maximum cache size, UltraLite returns an error message and the connection fails.

Any leading or trailing spaces in connection parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

#### Example

The following connection string fragment sets the minimum cache size to 1 MB.

```
"CACHE_MIN_SIZE=1m"
```

## Related Information

- [UltraLite Connection Strings and Parameters \[page 39\]](#)
- [UltraLite Database Connections \[page 647\]](#)
- [UltraLite Database Connection Using Embedded SQL \[page 682\]](#)
- [Connection Setup for an UltraLite Database \[page 602\]](#)
- [UltraLite CACHE\\_MAX\\_SIZE Connection Parameter \[page 185\]](#)
- [UltraLite CACHE\\_SIZE Connection Parameter \[page 187\]](#)
- [UltraLite cache\\_allocation Option \[page 207\]](#)
- [UltraLite page\\_size Creation Option \[page 166\]](#)
- [UltraLite RESERVE\\_SIZE Connection Parameter \[page 199\]](#)

### 1.12.2.4 UltraLite CACHE\_SIZE Connection Parameter

Defines the initial size of the database cache. UltraLite manages the cache size automatically, so setting this parameter should not be necessary.

For Android devices, you can use `Configuration.setPageSize` as an alternative to setting this connection parameter.

#### ☰, Syntax

```
CACHE_SIZE=number { k | m }
```

## Default

The default initial cache size is determined by the amount of memory available on your system and the size of the database.

## Remarks

The `cache_size` connection parameter specifies the initial amount of memory to allocate for the file cache. This cache is used to hold recently used pages from the database file in memory so they can be accessed quickly when needed again, and also to collect multiple modifications to a page before writing it back to storage. Accessing a page from the cache is many times faster than reading from storage. Writing to storage is more expensive, so grouping multiple modifications in a single write is important for performance. Encrypted databases also benefit from the cache because decryption occurs only when the page is loaded into the cache, and encryption occurs before the page is written back to storage. If the cache is sufficiently large, the overhead of encryption becomes negligible.

As an example of cache usage, consider synchronization. While UltraLite is receiving a download, the rows are inserted into the database, and referential integrity checks are performed. When inserted, the rows are also

indexed; they are added to each index on the table. So, while synchronizing, the cache tends to hold the pages where the new rows are stored, as well as the index pages for the current table. Synchronization performance depends on whether the cache is large enough to contain an appropriate working set of pages for a table being synchronized. If the cache is too small, row inserts may require repeated reads of index pages from storage, incurring a noticeable performance penalty over the case when the required index pages fit in the cache.

By default, the size is in bytes. Use k or m to specify units of kilobytes or megabytes.

If you exceed the permissible maximum cache size, it is automatically replaced with your platform's upper cache size limit.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

## Example

The following connection string fragment sets the cache size to 20 MB.

```
"CACHE_SIZE=20m"
```

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite CACHE\\_MAX\\_SIZE Connection Parameter \[page 185\]](#)

[UltraLite CACHE\\_MIN\\_SIZE Connection Parameter \[page 186\]](#)

[UltraLite cache\\_allocation Option \[page 207\]](#)

[UltraLite page\\_size Creation Option \[page 166\]](#)

[UltraLite RESERVE\\_SIZE Connection Parameter \[page 199\]](#)

### 1.12.2.5 UltraLite COMMIT\_FLUSH Connection Parameter

Determines when committed transactions are flushed to storage after a commit call. If no calls to commit are made by the UltraLite application, no flush can occur.

☞ Syntax

```
COMMIT_FLUSH={ immediate | grouped | on_checkpoint }
```

## Default

immediate

## Remarks

This connection parameter defines which transactions are recovered following a hardware failure or crash. You can group logical autocommit operations as a single recovery point.

By grouping these operations, you can improve UltraLite performance, but at the expense of data recoverability. There is a slight chance that a transaction may be lost, even though it has been committed, if a hardware failure or crash occurs after a commit, but before the transaction is flushed to storage.

The following parameters are supported:

### **immediate**

Committed transactions are flushed to storage immediately upon a commit call before the commit operation completes.

### **grouped**

Committed transactions are flushed to storage on a commit call, but only after a threshold you configure has been reached. You can configure either a transaction count threshold with the `commit_flush_count` database option or a time-based threshold with the `commit_flush_timeout` database option.

If set, both the `commit_flush_count` and the `commit_flush_timeout` options act as possible triggers for the commit flush; the first threshold that is met triggers the flush. When the flush occurs, UltraLite sets the counter and the timer back to 0. Then, both the counter and timer are monitored, until one of these thresholds is reached again.

### **on\_checkpoint**

Committed transactions are flushed to storage on a checkpoint operation. You can perform a checkpoint with any of the following:

- The CHECKPOINT statement. APIs that do not have a checkpoint method must use this SQL statement.
- The ULCheckpoint function for UltraLite Embedded SQL.
- The Checkpoint method on a connection object in a C++ component.

## Related Information

[Flush Single or Grouped Transactions \[page 586\]](#)

[UltraLite commit\\_flush\\_count Option \[Temporary\] \[page 208\]](#)

[UltraLite commit\\_flush\\_timeout Option \[Temporary\] \[page 209\]](#)

[CHECKPOINT Statement \[UltraLite\] \[page 529\]](#)

## 1.12.2.6 UltraLite CON Connection Parameter

Names a connection so that switching to it is easier in multi-connection applications.

☞ Syntax

```
CON=name
```

### Default

No connection name.

### Remarks

The CON connection parameter is global to the application.

Do not use this connection parameter unless you are going to establish and switch between two or more concurrent connections.

The connection name is not the same as the database name.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

### Example

The following connection string fragment sets the first connection name to MyFirstCon.

```
"CON=MyFirstCon"
```

### Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite DBN Connection Parameter \[page 193\]](#)

## 1.12.2.7 UltraLite DBF Connection Parameter

Specify the path and file name for an UltraLite database.

Use this connection parameter to specify the path and file name for a new database file or when connecting to an existing database file.

☰ Syntax

```
DBF=u1-db
```

### Behavior

1. On connect, look to see if the database is already running. If DBN is specified, look for a matching database and connect if found, proceed to auto-start if not.
2. If DBF is specified, look for a matching database (identical filename) and connect if found, proceed to auto-start if not.
3. If neither DBN nor DBF is specified, and a single database is running, connect to it.
4. A database is auto-started when required if DBF is specified. If DBN is also specified, it becomes the name of the running database, otherwise a name is generated from the base filename.

### Remarks

If you are connecting to multiple databases on different devices from a single connection string, you can use the following parameters to name platform-specific alternates:

- desktop:DBF
- device:DBF

If specified, these platform-specific connection parameters take precedence over DBF.

The value of DBF must meet the file name requirements for the platform.

#### Microsoft Windows Mobile

If you are deploying to a Microsoft Windows Mobile device, UltraLite administration tools running on the Microsoft Windows desktop can connect to an UltraLite database on an attached Microsoft Windows Mobile device. To identify a file on a Microsoft Windows Mobile device, you must specify the required absolute path, and use the [wce:](#) file prefix.

You cannot use the [wce:](#) file prefix in an application running on the Microsoft Windows Mobile device.

Any leading or trailing spaces in parameter values are ignored. The value cannot include leading single quotes, leading double quotes, or semicolons.

## Example

To connect to the database, `MyULdb.udb`, installed in the desktop directory `c:\Databases`, use the following connection string:

```
"DBF=c:\Databases\MyULdb.udb"
```

Note that this file path is Microsoft Windows desktop-specific and not appropriate for a mobile device.

The following example illustrates how to connect from an application to the `MyULdb.udb` database using platform-specific file paths:

```
"desktop:DBF=c:\databases\MyULdb.udb;device:DBF=\Documents\MyULdb.udb"
```

When the application is running on the desktop platform, the `desktop:DBF` connection parameter is used. When the application is running on the mobile device, the `device:DBF` connection parameter is used.

To connect from the desktop (using an administration tool) to the `MyULdb.udb` database that is deployed to the `Documents` folder of the attached Microsoft Windows Mobile device, use the following connection string:

```
"DBF=wce:\Documents\MyULdb.udb"
```

Connections to databases on mobile devices from the desktop is supported for Microsoft Windows/Microsoft Windows Mobile only.

## Related Information

- [UltraLite Connection Strings and Parameters \[page 39\]](#)
- [UltraLite File Path Formats in Connection Parameters \[page 42\]](#)
- [Precedence of Connection Parameters for UltraLite Administration Tools \[page 41\]](#)
- [UltraLite Database Connections \[page 647\]](#)
- [UltraLite Database Connection Using Embedded SQL \[page 682\]](#)
- [Connection Setup for an UltraLite Database \[page 602\]](#)
- [UltraLite DBN Connection Parameter \[page 193\]](#)

### 1.12.2.8 UltraLite DBKEY Connection Parameter

When creating a new UltraLite database, this connection parameter provides an encryption key for the database.

When opening a connection to an existing database, it provides the encryption key for the database.

☞ Syntax

```
DBKEY=string
```



## Default

No key is provided.

## Remarks

If you do not specify the correct encryption key for the database, the connection fails.

If a database is created using an encryption key, the database file is strongly encrypted by using either the 256-bit AES or FIPS-certified 256-bit AES algorithm. By using strong encryption, you have increased security against skilled and determined attempts to gain access to the data.

Any leading or trailing spaces in parameter values are ignored. The value cannot include leading single quotes, leading double quotes, or semicolons.

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[Database Security \[page 35\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite obfuscate Creation Option \[page 165\]](#)

## 1.12.2.9 UltraLite DBN Connection Parameter

Differentiates databases by name when applications connect to more than one database.

≡, Syntax

```
DBN=db-name
```

## Default

None.

## Behavior

1. On connect, look to see if the database is already running. If DBN is specified, look for a matching database and connect if found, proceed to auto-start if not.
2. If DBF is specified, look for a matching database (identical filename) and connect if found, proceed to auto-start if not.
3. If neither DBN nor DBF is specified, and a single database is running, connect to it.
4. A database is auto-started when required if DBF is specified. If DBN is also specified, it becomes the name of the running database, otherwise a name is generated from the base filename.

## Remarks

UltraLite sets the database name after the database has been opened. Client applications can then connect to this database via its name instead of its file.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

## Example

Use the following parameters to connect to the running UltraLite database named Kitchener:

```
DBN=Kitchener;DBF=cities.udb
```

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite DBF Connection Parameter \[page 191\]](#)

### 1.12.2.10 UltraLite Desktop Connection Parameter Prefix

Use the prefix `desktop:` with a connection parameter to indicate that the parameter only applies when the application is running on the desktop.

If the connection parameter is appropriate for both desktop and mobile device, then do not use the prefix.

### ☰, Syntax

```
desktop:connection-parameter=value
```

## Remarks

Use the `desktop` connection parameter prefix for UltraLite client applications that run on a variety of devices.

Connection parameters with a `desktop` or `device` prefix take precedence over parameters without a prefix.

## Example

The following example identifies different database files for the desktop and the mobile device, the location of the temporary directory on the desktop, and the `cache_size` for the mobile device:

```
"desktop:DBF=C:\dir\db.udb;device:DBF=\SD Card\db.udb;desktop:temp_dir=\Temp;device:cache_size=4M"
```

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite File Path Formats in Connection Parameters \[page 42\]](#)

[Precedence of Connection Parameters for UltraLite Administration Tools \[page 41\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite DBF Connection Parameter \[page 191\]](#)

### 1.12.2.11 UltraLite Device Connection Parameter Prefix

Use the prefix `device:` with a connection parameter to indicate that the parameter only applies when the application is running on the mobile device.

If the connection parameter is appropriate for both desktop and mobile device, then do not use the prefix.

### ☰, Syntax

```
device:connection-parameter=value
```

## Remarks

Use the `device` connection parameter prefix for UltraLite client applications that run on a variety of devices. Connection parameters with a `desktop` or `device` prefix take precedence over parameters without a prefix.

## Example

The following example identifies different database files for the desktop and the mobile device, the location of the temporary directory on the mobile device, and the `cache_size` for the mobile device:

```
"desktop:DBF=C:\dir\db.udb;device:DBF=\SD Card\db.udb;device:temp_dir=\Temp;device:cache_size=4M"
```

## Related Information

- [UltraLite Connection Strings and Parameters \[page 39\]](#)
- [UltraLite File Path Formats in Connection Parameters \[page 42\]](#)
- [Precedence of Connection Parameters for UltraLite Administration Tools \[page 41\]](#)
- [UltraLite Database Connections \[page 647\]](#)
- [UltraLite Database Connection Using Embedded SQL \[page 682\]](#)
- [Connection Setup for an UltraLite Database \[page 602\]](#)
- [UltraLite DBF Connection Parameter \[page 191\]](#)

### 1.12.2.12 UltraLite MIRROR\_FILE Connection Parameter

Specifies the name of the database mirror file to which all database writes are to be issued (at the same time as they are to the main database file).

≡ Syntax

```
MIRROR_FILE=path\mirrorfile-db
```

## Default

None.

## Remarks

UltraLite provides basic database file mirroring to improve fault tolerance on potentially unreliable storage systems. This is accomplished using the mirror file. All database writes are issued to the mirror file at the same time as they are to the main database file (write overhead is therefore doubled; read overhead is not affected). If a corrupt page is read from the database file, the page is recovered by reading from the mirror file.

Mirroring is supported on all platforms using a file-based store.

When the `mirror_file` option is specified when you start the database, UltraLite will open the named file and verify that it matches the main database file before continuing. If the mirror file does not exist, it is created at that point by copying the main file. If the mirror is not a database file, or is corrupt, an error is reported and the database will not start until the file is removed or a different mirror is specified. If the mirror does not match the database, `SQL_E_MIRROR_FILE_MISMATCH` is generated and the database will not start. When a corrupt page is recovered, the warning `SQL_E_CORRUPT_PAGE_READ_RETRY` is generated. (Without mirroring, or if the mirror file is also corrupt, the error `SQL_E_DEVICE_ERROR` is generated and the database is halted.)

To effectively protect against media failures, page checksums must be enabled when you use a mirror file. (With or without mirroring, page checksums allow UltraLite to detect page corruption as soon as the page is loaded and avoid referencing corrupt data.) Starting with version 17.0.10, checksum validation is enabled by default. The `checksum_level` database creation option controls checksum validation. UltraLite will generate the warning `SQL_E_MIRROR_FILE_REQUIRES_CHECKSUMS` if checksums are not enabled when using a mirror file.

Because the mirror is an exact copy of the database file, it can be started directly as a database. The `ulvalid` utility reports corrupt pages.

## Example

The following example creates a new connection and creates a mirror file:

```
Connection = DatabaseMgr.OpenConnection("DBF=c:\Dbfile.udb; UID=JDoe;PWD=ULdb;  
MIRROR_FILE=c:\test\MyMirrorDB.udb")
```

## Related Information

- [UltraLite Connection Strings and Parameters \[page 39\]](#)
- [UltraLite File Path Formats in Connection Parameters \[page 42\]](#)
- [Precedence of Connection Parameters for UltraLite Administration Tools \[page 41\]](#)
- [UltraLite Database Connections \[page 647\]](#)
- [UltraLite Database Connection Using Embedded SQL \[page 682\]](#)
- [Connection Setup for an UltraLite Database \[page 602\]](#)
- [UltraLite `checksum\_level` Creation Option \[page 149\]](#)
- [UltraLite Validate Database Utility \(`ulvalid`\) \[page 246\]](#)

## 1.12.2.13 UltraLite PWD Connection Parameter

When creating a new UltraLite database, this connection parameter sets the password for the default user.

When connecting to an existing database, it defines the password for a user ID that is used for authentication.

For Android devices, you can use `Configuration.setPassword` as an alternative to setting this connection parameter.

☰ Syntax

```
PWD=password
```

### Default

If you do not set both the UID and PWD, UltraLite opens connections with `UID=DBA` and `PWD=sql`.

### Remarks

Every user of a database has a password. UltraLite supports up to four user ID/password combinations.

You can set passwords to NULL or an empty string.

A random 4-byte salt value is generated when a new user is created or an existing user changes their password. The salt value is appended to the user's password when calculating the password hash and is stored in the database along with the hash. Salting significantly decreases vulnerability to dictionary attacks and also ensures that users with the same password will have different password hashes.

This connection parameter is not encrypted. However, UltraLite hashes the password before saving it, so you can only modify a password using SQL Central.

### Example

The following partial connection string supplies the user ID DBA and password sql:

```
"UID=DBA;PWD=sql"
```

The following partial connection string supplies the user ID DBA and an empty password:

```
"UID=DBA;PWD=''"
```

## Related Information

[Users](#)

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Users \[page 66\]](#)

[User Authentication \[page 697\]](#)

[UltraLite Database Connections \[page 647\]](#)

[User Authentication \[page 697\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite UID Connection Parameter \[page 202\]](#)

### 1.12.2.14 UltraLite RESERVE\_SIZE Connection Parameter

Pre-allocates the file system space required for your UltraLite database, without actually inserting any data. Reserving the file system space prevents the space from being used up by other files.

≡ Syntax

```
RESERVE_SIZE= number{ k | m | g }
```

#### Default

0 (no reserve size).

#### Remarks

The value you supply can be any value from 0 to your maximum database size. Use k, m, or g to specify units of kilobytes, megabytes, or gigabytes, respectively. If you do not specify a unit, bytes are assumed by default.

You should run the database with test data and observe the database size and choose a reserve size that suits your UltraLite deployment.

If the RESERVE\_SIZE value is smaller than the database size, UltraLite ignores the parameter.

Reserving file system space can improve performance slightly because it may:

- Reduce the degree of file fragmentation compared to growing incrementally.
- Prevent out-of-storage memory failures.

Because an UltraLite database consists of data and metadata, the database size grows only when required (when the application updates the database).

## Example

The following connection string fragment sets the reserve size to 128 KB so the system reserves that much system space for the database upon startup.

```
"RESERVE_SIZE=128K"
```

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite CACHE\\_SIZE Connection Parameter \[page 187\]](#)

[UltraLite page\\_size Creation Option \[page 166\]](#)

### 1.12.2.15 UltraLite START Connection Parameter

Starts the UltraLite engine executable. This parameter is not supported for UltraLite for Android. This parameter is only required if the engine is not in one of the expected locations.

☰ Syntax

```
START=path\uleng17
```

## Remarks

Only supply a StartLine (START) connection parameter if you are connecting to an engine that is not currently running.

Paths with spaces require quotes. Otherwise, the client returns SQLE\_UNABLE\_TO\_CONNECT\_OR\_START.

## Example

The following command starts the UltraLite engine that is located in the Program Files directory:

```
Start="\Program Files\uleng17.exe"
```



## Related Information

[UltraLite Engine Startup \[page 133\]](#)

[UltraLite Data Management Components for Microsoft Windows Mobile \[page 21\]](#)

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

[UltraLite Engine Utility \(uleng17\) \[page 223\]](#)

### 1.12.2.16 UltraLite TEMP\_DIR Connection Parameter

Specifies the name of the directory (which must already exist) into which UltraLite will place the temporary file (with a name derived from the database name).

☰ Syntax

```
TEMP_DIR=path
```

#### Remarks

In addition to the database file, UltraLite creates and maintains a temporary file during database operation. You do not need to work with or maintain the file in any way.

By default, UltraLite maintains its temporary file in the same folder (if one exists) as the UltraLite database itself. The temporary file has the same file name as the database, but for file-based platforms the tilde is included in the extension of the file. For example, if you run the `CustDB.udb` sample database, the temporary file called `CustDB.~db` is maintained in the same directory as the database file.

Specifying a temporary directory with faster I/O characteristics can improve the performance of things like temporary tables which are large relative to the cache size. Long-running transactions can also consume noticeable space in the temp file.

Paths with spaces require quotes. Otherwise, the client returns `SQLE_UNABLE_TO_CONNECT_OR_START`.

#### Example

The following connection string fragment puts the temporary file in the `\Temp` directory:

```
temp_dir=\Temp;
```

## Related Information

[UltraLite Database Connections \[page 647\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

### 1.12.2.17 UltraLite UID Connection Parameter

When creating a new UltraLite database, this connection parameter sets the default user ID for the database.

When connecting to an existing database, it specifies the user ID with which you connect to the database. The value must be an authenticated user for the database.

☞ Syntax

```
UID=user
```

#### Default

If you do not set the UID and PWD when connecting, UltraLite opens connections with *UID=DBA* and *PWD=sql*.

#### Remarks

Every user of a database has a user ID. UltraLite supports up to four user ID/password combinations.

UltraLite user IDs are separate from MobiLink user names and from other SQL Anywhere user IDs. You cannot change a user ID once it is created. Instead, you must delete the user ID and then add a new one.

You cannot set the UID to NULL or an empty string. The maximum length for a user ID is 31 characters. User IDs are case insensitive.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

#### Example

The following connection string fragment supplies the user ID DBA and password sql for a database:

```
"UID=DBA;PWD=sql"
```

## Related Information

[UltraLite Connection Strings and Parameters \[page 39\]](#)

[UltraLite Users \[page 66\]](#)

[User Authentication \[page 697\]](#)

[UltraLite Database Connections \[page 647\]](#)

[User Authentication \[page 697\]](#)

[UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

[Connection Setup for an UltraLite Database \[page 602\]](#)

### 1.12.3 UltraLite Database Properties

UltraLite database property values are defined when the database is first created.

Properties can be changed by re-creating the UltraLite database or editing their corresponding database option, if available.

UltraLite supports the following database properties:

Property	Description
<i>cache_allocation</i>	Returns the current cache size as a percentage of the minimum and maximum settings. This property corresponds to the <code>cache_allocation</code> option for the database.
<i>CaseSensitive</i>	Returns the status of the case sensitivity feature. Returns On if the database is case sensitive. Otherwise, it returns Off.
<i>CharSet</i>	Returns the CHAR character set of the database. The character set used by the database is determined by the database's collation sequence and whether the data is UTF-8 encoded.
<i>ChecksumLevel</i>	Returns the level of checksum validation in the database, one of 0 (do not add checksums), 1 (add checksums only to important pages), or 2 (add checksums to all pages). The default value is 2. This property corresponds to the <code>checksum_level</code> creation option for the database.
<i>Collation</i>	Returns the name of the database's collation sequence. This property corresponds to the collation creation option for the database.
<i>commit_flush_count</i>	Returns the value of the <code>commit_flush_count</code> option that sets a commit count threshold. This property corresponds to the <code>commit_flush_count</code> option [temporary] for the database.
<i>commit_flush_timeout</i>	Returns the value of the <code>commit_flush_timeout</code> option that sets a time interval threshold. This property corresponds to the <code>commit_flush_timeout</code> option [temporary] for the database.

Property	Description
<i>ConnCount</i>	Returns the number of connections to the database. The value is dynamic: it can vary depending on how many connections currently exist. UltraLite supports up to fourteen concurrent database connections.
<i>date_format</i>	Returns the date format the database uses for string conversions. This property corresponds to the <code>date_format</code> creation option for the database.
<i>date_order</i>	Returns the date order the database uses for string conversions. This property corresponds to the <code>date_order</code> creation option for the database.
<i>Encryption</i>	<p>Returns the type of database encoding, one of None, Simple, AES, or AES_FIPS.</p> <p>The encoding used by the database is determined by whether you have configured strong encryption (AES or AES_FIPS) and the <code>DBKEY</code> creation parameter, or simple obfuscation.</p> <p>The only time this property can change is when the value is originally None (that is, neither fips nor obfuscation is used) and you then change the encryption key by specifying a new encryption key on the Connection object by calling the correct function or method for your API. In this case, the value would change to AES because the fips creation parameter cannot be set after the database has been created.</p> <p>You can use API methods to change the encryption key.</p>
<i>File</i>	Returns the name of the database root file for the current connection, including the path. This is the value specified in the <code>DBF</code> connection parameter value.
<i>global_database_id</i>	Returns the value of the <code>global_database_id</code> option used for global autoincrement columns. This property corresponds to the <code>global_database_id</code> option for the database.
<i>isolation_level</i>	Returns the current isolation level of the database. The value can either be <i>read_committed</i> or <i>read_uncommitted</i> . This property corresponds to the <code>isolation_level</code> option for the database.
<i>MaxHashSize</i>	Returns the default number of maximum bytes to use for index hashing. This property can be set on a per-index basis. This property corresponds to the <code>max_hash_size</code> creation option for the database.
<i>ml_remote_id</i>	Returns the value of the <code>ml_remote_id</code> option that uniquely identifies the database for MobiLink synchronization. This property corresponds to the <code>ml_remote_id</code> option for the database.

Property	Description
<i>Name</i>	Returns the name (or alias) of the database for the current connection. The name returned matches the DBN connection parameter value. If you did not use the DBN connection parameter, the name returned is the database file without the path and extension.
<i>nearest_century</i>	Returns the nearest century the database uses for string conversions. This property corresponds to the nearest_century creation option for this database.
<i>PageSize</i>	Returns the page size of the database, in bytes. This property corresponds to the page_size creation option for the database.
<i>PartialDownload</i>	Returns Yes or No to indicate whether the database contains a partial download.
<i>precision</i>	Returns the floating-point precision the database uses for string conversions. This property corresponds to the precision creation option for the database.
<i>scale</i>	Returns the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum <i>precision</i> during string conversions by the database. This property corresponds to the scale creation option for the database.
<i>time_format</i>	Returns the time format the database uses for string conversions. This property corresponds to the time_format creation option for the database.
<i>timestamp_format</i>	Returns the timestamp format the database uses for string conversions. This property corresponds to the timestamp_format creation option for the database.
<i>timestamp_increment</i>	Returns the minimum difference between two unique timestamps, in microseconds. This property corresponds to the timestamp_increment creation option for the database.
<i>timestamp_with_time_zone_format</i>	Returns the timestamp format for TIMESTAMP WITH TIME ZONE values. This property corresponds to the timestamp_with_time_zone_format creation option for the database.
<i>UploadUnknown</i>	Returns true if the last synchronization failed after the upload was sent but before the upload acknowledgement from the synchronization server was received. When this property is true, on the next synchronization UltraLite first asks MobiLink if it received its last upload before continuing with the new synchronization.

## Related Information

[UltraLite Database Options \[page 206\]](#)

[Isolation Levels \[page 48\]](#)

[Database Security \[page 35\]](#)  
[Reading Database Properties \[page 43\]](#)  
[Accessing Database Options \[page 45\]](#)  
[UltraLite Options \[page 143\]](#)  
[UltraLite cache\\_allocation Option \[page 207\]](#)  
[UltraLite case Creation Option \[page 147\]](#)  
[UltraLite utf8\\_encoding Creation Option \[page 180\]](#)  
[UltraLite collation Creation Option \[page 150\]](#)  
[UltraLite checksum\\_level Creation Option \[page 149\]](#)  
[UltraLite collation Creation Option \[page 150\]](#)  
[UltraLite commit\\_flush\\_count Option \[Temporary\] \[page 208\]](#)  
[UltraLite commit\\_flush\\_timeout Option \[Temporary\] \[page 209\]](#)  
[UltraLite date\\_format Creation Option \[page 151\]](#)  
[UltraLite date\\_order Creation Option \[page 153\]](#)  
[UltraLite DBF Connection Parameter \[page 191\]](#)  
[UltraLite global\\_database\\_id Option \[page 210\]](#)  
[UltraLite max\\_hash\\_size Creation Option \[page 161\]](#)  
[UltraLite ml\\_remote\\_id Option \[page 211\]](#)  
[UltraLite DBN Connection Parameter \[page 193\]](#)  
[UltraLite DBF Connection Parameter \[page 191\]](#)  
[UltraLite nearest\\_century Creation Option \[page 162\]](#)  
[UltraLite page\\_size Creation Option \[page 166\]](#)  
[UltraLite Precision Creation Option \[page 168\]](#)  
[UltraLite scale Creation Option \[page 169\]](#)  
[UltraLite time\\_format Creation Option \[page 171\]](#)  
[UltraLite timestamp\\_format Creation Option \[page 173\]](#)  
[UltraLite timestamp\\_increment Creation Option \[page 175\]](#)  
[UltraLite timestamp\\_with\\_time\\_zone\\_format Creation Option \[page 177\]](#)  
[UltraLite fips Creation Option \[page 159\]](#)  
[UltraLite obfuscate Creation Option \[page 165\]](#)  
[UltraLite DBKEY Connection Parameter \[page 192\]](#)

## 1.12.4 UltraLite Database Options

UltraLite database option values are defined when the database is first created and can be altered while connected to the database.

### In this section:

[UltraLite cache\\_allocation Option \[page 207\]](#)

Explicitly resizes the cache. The value is a percentage of the minimum-to-maximum range. A value of zero means the minimum size, and a value of 100 means the maximum size.

[UltraLite commit\\_flush\\_count Option \[Temporary\] \[page 208\]](#)

Sets a commit count threshold, after which a commit flush is performed.

[UltraLite commit\\_flush\\_timeout Option \[Temporary\] \[page 209\]](#)

Sets a time interval threshold, after which a grouped commit flush is performed.

[UltraLite global\\_database\\_id Option \[page 210\]](#)

Sets the database identification number.

[UltraLite isolation\\_level Option \[page 211\]](#)

Isolation levels define the degree to which the operations in one transaction are visible to the operations in other concurrent transactions. UltraLite uses the default isolation level, read\_committed, for connections in auto-commit mode.

[UltraLite ml\\_remote\\_id Option \[page 211\]](#)

The **remote ID** is a unique identifier for an UltraLite database that is used by MobiLink to identify the database for synchronization.

## Related Information

[Accessing Database Options \[page 45\]](#)

[Reading Database Properties \[page 43\]](#)

[UltraLite Options \[page 143\]](#)

### 1.12.4.1 UltraLite cache\_allocation Option

Explicitly resizes the cache. The value is a percentage of the minimum-to-maximum range. A value of zero means the minimum size, and a value of 100 means the maximum size.

#### Allowed Values

Integer

#### Default

None.

#### Remarks

The cache\_allocation property returns the current cache size as a percentage value of the minimum and maximum cache size.

## Related Information

[Accessing Database Options \[page 45\]](#)

[UltraLite Database Properties \[page 203\]](#)

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

[UltraLite CACHE\\_SIZE Connection Parameter \[page 187\]](#)

[UltraLite CACHE\\_MIN\\_SIZE Connection Parameter \[page 186\]](#)

[UltraLite CACHE\\_MAX\\_SIZE Connection Parameter \[page 185\]](#)

### 1.12.4.2 UltraLite commit\_flush\_count Option [Temporary]

Sets a commit count threshold, after which a commit flush is performed.

#### Allowed Values

Integer

#### Default

10

#### Remarks

Use 0 to disable the transaction count. When the transaction count is disabled, the number of commits is unlimited when a flush is triggered.

Both `commit_flush_count` and `commit_flush_timeout` are temporary database options. You must set these options each time you start a database. They persist as long as the database continues to run. They are only required when you set `COMMIT_FLUSH=grouped` as part of a connection string.

When you set this option and set the `COMMIT_FLUSH` connection parameter to `grouped` in your connection string, either threshold triggers a flush. When the flush occurs, UltraLite sets the counter *and* the timer back to 0. Then, both the counter and timer are monitored until one of these thresholds is subsequently reached.

An important consideration for setting the commit flush options is how much the delay to flush committed transactions poses a risk to the recoverability of your data. There is a slight chance that a transaction may be lost, even though it has been committed. If a serious hardware failure occurs after a commit, but before the transaction is flushed to storage, the transaction is rolled back on recovery. A longer delay can increase UltraLite performance. You must choose an appropriate count threshold with care.



To set the `commit_flush_count` option from a client application, set the option using the set database option function for the programming interface you are using or use the SET OPTION SQL statement.

## Related Information

[Flush Single or Grouped Transactions \[page 586\]](#)

[Accessing Database Options \[page 45\]](#)

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

[UltraLite `commit\_flush\_timeout` Option \[Temporary\] \[page 209\]](#)

[UltraLite COMMIT\\_FLUSH Connection Parameter \[page 188\]](#)

### 1.12.4.3 UltraLite `commit_flush_timeout` Option [Temporary]

Sets a time interval threshold, after which a grouped commit flush is performed.

#### Allowed Values

Integer, in milliseconds

#### Default

10000 milliseconds

#### Remarks

Use 0 to disable the time threshold.

Both `commit_flush_count` and `commit_flush_timeout` are temporary database options. You must set these options each time you start a database. They persist as long as the database continues to run. They are only required when you set `COMMIT_FLUSH=grouped` as part of a connection string.

If you set this option in addition to the `commit_flush_timeout` option and if you have set the `COMMIT_FLUSH` connection parameter to `grouped`, either threshold triggers a flush. When the flush occurs, UltraLite sets the counter *and* the timer back to 0. Then, both the counter and timer are monitored until one of these thresholds is subsequently reached.

An important consideration for setting the commit flush options is how much the delay to flush committed transactions poses a risk to the recoverability of your data. There is a slight chance that a transaction may be lost, even though it has been committed. If a serious hardware failure occurs after a commit, but before the

transaction is flushed to storage, the transaction is rolled back on recovery. A longer delay can increase UltraLite performance. You must choose an appropriate timeout threshold with care.

To set the `commit_flush_timeout` option from a client application, set it using the set database option function for the programming interface you are using or use the SET OPTION SQL statement.

## Related Information

[Accessing Database Options \[page 45\]](#)

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

[UltraLite `commit\_flush\_count` Option \[Temporary\] \[page 208\]](#)

[UltraLite `COMMIT\_FLUSH` Connection Parameter \[page 188\]](#)

### 1.12.4.4 UltraLite `global_database_id` Option

Sets the database identification number.

#### Allowed Values

Unique, non-negative integer

#### Default

The range of default values for a particular global autoincrement column is  $p_n + 1$  to  $p(n + 1)$ , where  $p$  is the partition size of the column and  $n$  is the global database identification number.

#### Remarks

To maintain primary key uniqueness when synchronizing with a MobiLink server, the global ID sets a starting value for GLOBAL AUTOINCREMENT columns. The global ID must be set before default values can be assigned. If a row is added to a table and does not have a value set already, UltraLite generates a value for the column by combining the `global_database_id` value and the partition size.

When this option is set, UltraLite performs a commit.

When deploying an application, you must assign a different identification number to each database for synchronization with the MobiLink server. You can change the global ID of an existing database at any time.

To set the `global_database_id` option from a client application, use the set database option function for the programming interface you are using or use the SET OPTION SQL statement.

## Example

To autoincrement UltraLite database columns from 3001 to 4000, set the global ID to 3.

```
SET OPTION global_database_id=3
```

## Related Information

[GLOBAL AUTOINCREMENT](#)

[GLOBAL AUTOINCREMENT Columns in UltraLite \[page 75\]](#)

[Accessing Database Options \[page 45\]](#)

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

### 1.12.4.5 UltraLite isolation\_level Option

Isolation levels define the degree to which the operations in one transaction are visible to the operations in other concurrent transactions. UltraLite uses the default isolation level, `read_committed`, for connections in auto-commit mode.

## Related Information

[Isolation Levels \[page 48\]](#)

[Accessing Database Options \[page 45\]](#)

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

### 1.12.4.6 UltraLite ml\_remote\_id Option

The **remote ID** is a unique identifier for an UltraLite database that is used by MobiLink to identify the database for synchronization.

The remote ID can be any string that has meaning to you, as long as the string remains unique among all remote MobiLink clients. The ID can also be set to NULL (NULL is the initial value). During synchronization, if the remote ID is NULL, UltraLite will assign it to a generated GUID.

If you prepopulate an UltraLite database using synchronization for distribution to multiple devices, you must reset the remote ID to NULL before distribution to ensure that each database has a unique remote ID. Upon distribution, a new unique remote ID can be set explicitly or it can be left as NULL so that UltraLite will automatically generate a new unique value.

## Allowed Values

Any value that uniquely identifies the database for MobiLink synchronization.

## Default

Null

## Remarks

MobiLink uses the remote ID to store the synchronization information for the remote database. Given the remote ID, MobiLink user names are no longer required to be unique. The remote ID becomes particularly useful when you have multiple MobiLink users synchronizing the same UltraLite database. In this case, your synchronization scripts should reference the remote ID and not just the user name.

When this option is set, UltraLite performs a commit.

To set the `ml_remote_id` option from a client application, set it using the set database option function for the programming interface you are using or use the SET OPTION SQL statement.

## Related Information

[Remote IDs](#)

[Accessing Database Options \[page 45\]](#)

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

[User Name Synchronization Parameter \[page 120\]](#)

## 1.12.5 UltraLite Utilities

UltraLite provides utilities that are designed to perform basic database administration activities at a command prompt. Many of these utilities share a similar functionality to the SQL Anywhere Server utilities. However, the way options are used can vary.

Always refer to the UltraLite reference documentation for the UltraLite implementation of these options.

### **i** Note

Options for the utilities documented are case sensitive, unless otherwise noted. Type options *exactly* as they are displayed.

## In this section:

### [Supported Exit Codes \[page 214\]](#)

The ulload, ulsync, and ulunload utilities return exit codes to indicate whether the operation a utility attempted to complete was successful. 0 indicates a successful operation. Any other value indicates that the operation failed.

### [Interactive SQL for UltraLite Utility \(dbisql\) \[page 214\]](#)

Executes SQL statements and runs script files against a database.

### [SQL preprocessor for UltraLite Utility \(sqlpp\) \[page 219\]](#)

Preprocesses a C/C++ program that contains Embedded SQL (ESQL), so that code required for that program can be generated before you run the compiler.

### [UltraLite Engine Utility \(uleng17\) \[page 223\]](#)

Manages concurrent UltraLite database connections from applications, and allows the UltraLite engine to run as a daemon using the -ud option.

### [UltraLite Engine Stop Utility \(ulstop\) \[page 224\]](#)

Stops the UltraLite engine.

### [UltraLite Erase Utility \(ulerase\) \[page 225\]](#)

Erases an UltraLite database.

### [UltraLite Information Utility \(ulinfo\) \[page 226\]](#)

Displays information about an UltraLite database.

### [UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

Creates a new UltraLite database.

### [UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

Loads data from an XML file into a new or existing database.

### [UltraLite Synchronization Utility \(ulsync\) \[page 238\]](#)

Synchronizes an UltraLite database with a MobiLink server. This tool can be used for testing synchronization during application development.

### [UltraLite Synchronization Profile Options \[page 241\]](#)

Specify synchronization profile options with the ulsync utility on the command line after you have defined all other command line options. Keywords are case insensitive.

### [UltraLite Database Unload Utility \(ulunload\) \[page 243\]](#)

Unloads either an entire UltraLite database to XML or SQL, or all or part of UltraLite data to XML or SQL.

### [UltraLite Validate Database Utility \(ulvalid\) \[page 246\]](#)

Performs a full (normal) validation of an UltraLite database.

## 1.12.5.1 Supported Exit Codes

The uload, ulsync, and ulunload utilities return exit codes to indicate whether the operation a utility attempted to complete was successful. 0 indicates a successful operation. Any other value indicates that the operation failed.

Exit code	Status	Description
0	EXIT_OKAY	Operation successful.
1	EXIT_FAIL	Operation failure.
3	EXIT_FILE_ERROR	Database cannot be found.
4	EXIT_OUT_OF_MEMORY	Exhausted the dynamic memory of the device.
6	EXIT_COMMUNICATIONS_FAIL	Communications error generated while talking to the UltraLite engine.
9	EXIT_UNABLE_TO_CONNECT	Invalid UID or PWD provided, therefore cannot connect to the database.
12	EXIT_BAD_ENCRYPT_KEY	Missing or invalid encryption key.
13	EXIT_DB_VER_NEWER	Detected that the database version is incompatible. The database must be upgraded to a newer version.
255	EXIT_USAGE	Invalid command line options.

## 1.12.5.2 Interactive SQL for UltraLite Utility (dbisql)

Executes SQL statements and runs script files against a database.

### ≡ Syntax

```
dbisql -c "connection-string" [ options ] [ dbisql-statement | dbisql-script-file ]
```

```
dbisql -c "connection-string" -ul [ options ] [ dbisql-statement | dbisql-script-file ]
```

**dbisql-statement:** A SQL statement or a series of sql statements separated by a command-delimiter.

Option	Description
@data	<p>Reads options from the specified environment variable or configuration file.</p> <p>If both the environment variable and configuration file exist with the same name, the environment variable is used.</p> <p>To protect information in the configuration file, you can use the File Hiding utility (dbfhide) to encode the contents of the configuration file. Interactive SQL does not support configuration files that are encrypted.</p>
-c "keyword=value; ..."	<p>Specifies connection parameters. If Interactive SQL cannot connect, you are presented with a window where you can enter the connection parameters. If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.</p>
-d delimiter	<p>Specifies a command delimiter. Quotation marks around the delimiter are optional, but are required when the command shell itself interprets the delimiter in some special way.</p> <p>This option overrides the setting of the Interactive SQL command_delimiter option.</p>
-dl	<p>Echoes all statements explicitly executed by the user to the command window (STDOUT). This can provide useful feedback for debugging SQL scripts, or when Interactive SQL is processing a long SQL script. (The final character is a number 1, not a lowercase L). This option is only available when you run Interactive SQL as a command line program.</p>
-datasource DSN-name	<p>Specifies an ODBC data source to connect to.</p>

Option	Description
<code>-f filename</code>	<p>Opens (but does not run) the file called <code>filename</code> in the SQL Statements pane.</p> <p>If the <code>-f</code> option is given, the <code>-c</code> option is ignored; that is, no connection is made to the database.</p> <p>The file name can be enclosed in quotation marks, and <i>must</i> be enclosed in quotation marks if the file name contains a space.</p> <p>If the file does not exist, or if it is really a directory instead of a file, Interactive SQL prints an error message and then quits.</p> <p>If the file name does not include a full drive and path specification, it is assumed to be relative to the current directory.</p> <p>This option is only supported when Interactive SQL is run as a windowed application.</p>
<code>-host hostname</code>	<p>Specifies the <code>hostname</code> or IP address of the computer on which the database server is running. You can use the name <code>localhost</code> to represent the current computer.</p>
<code>-nogui</code>	<p>Runs Interactive SQL as a console application, with no windowed user interface. This is useful for batch operations.</p> <p>If you specify either <code>dbisql-statement</code> or <code>dbisql-script-file</code>, then <code>-nogui</code> is assumed.</p> <p>In this mode, Interactive SQL sets the program exit code to indicate success or failure. On Windows operating systems, the environment variable <code>ERRORLEVEL</code> is set to the program exit code.</p>
<code>-onerror { continue   exit }</code>	<p>Controls what happens if an error is encountered while reading statements from a script file. It is useful when using Interactive SQL in batch operations. This option overrides the Interactive SQL <code>on_error</code> option setting.</p> <p>Define one of the following supported <code>behavior</code> values:</p> <p><b>Continue</b></p> <p>The error is ignored and Interactive SQL continues executing statements.</p> <p><b>Exit</b></p> <p>Interactive SQL terminates.</p>



Option	Description
<code>-q</code>	<p>Suppresses output messages. Sets the utility to run in quiet mode. This is useful only if you start Interactive SQL with a statement or script file. Specifying this option does not suppress error messages, but it does suppress the following:</p> <ul style="list-style-type: none"> <li>warnings and other non-fatal messages</li> <li>the printing of result sets</li> </ul>
<code>-ul</code>	<p>Specifies that UltraLite databases are the default. Interactive SQL customizes the options available to you depending on the type of database you are connected to.</p> <p>By default, Interactive SQL assumes that you are connecting to SQL Anywhere databases. When you specify the <code>-ul</code> option, the default changes to UltraLite databases. Regardless of the type of database set as the default, you can connect to either SQL Anywhere or UltraLite databases by choosing the database type from the <i>Change Database Type</i> dropdown list on the <i>Connect</i> window.</p>
<code>-version</code>	<p>Displays the version number of Interactive SQL. You can also view the version number from within Interactive SQL; from the <i>Help</i> menu, click <i>About Interactive SQL</i>.</p>
<code>-x</code>	<p>Scans statements but does not execute them. This is useful for checking long script files for syntax errors.</p>
<code>dbisql-statement</code>   <code>dbisql-script-file</code>	<p>Execute the SQL statement or execute the specified <code>dbisql-script-file</code>.</p> <p>If you do not specify a <code>dbisql-statement</code> or <code>dbisql-script-file</code>, Interactive SQL enters interactive mode, where you can type a statement into a command window.</p>

## Remarks

Interactive SQL allows you to browse the database, execute SQL statements, and run script files. It also provides feedback about:

- the number of rows affected
- the time required for each statement
- the execution plan of queries
- any error messages

You can use Interactive SQL to connect to a SQL Anywhere database, an UltraLite database, an SAP IQ database, an SAP HANA database, or a generic ODBC database.

For Windows, there are two executables:

1. Batch scripts should call `dbisql` or `dbisql.com`, not `dbisql.exe`. The `dbisql.com` executable is linked as a console application.
2. The `dbisql.exe` executable is linked as a windowed application and does not block the command shell from which it was started. If `dbisql.exe` is run from a batch file, you won't see any output sent to the standard output or standard error files.

The default encoding for Interactive SQL can also be temporarily set using the Interactive SQL `default_isql_encoding` option.

You can specify the encoding to use when reading or writing files using the `ENCODING` clause of the `INPUT`, `OUTPUT`, or `READ` statement.

- `INPUT` statement
- `OUTPUT` statement
- `READ` statement

Exit codes are 0 (success) or non-zero (failure). Non-zero exit codes are set only when you run Interactive SQL in batch mode (with a command line that contains a SQL statement or the name of a script file).

In command-prompt mode, Interactive SQL sets the program exit code to indicate success or failure. On Windows operating systems, the environment variable `ERRORLEVEL` is set to the program exit code.

When executing a `reload.sql` file with Interactive SQL and the database is encrypted, you must specify the encryption key as a parameter. If you do not provide the key in the `READ` statement, Interactive SQL prompts for the key.

You can start Interactive SQL in the following ways:

- From SQL Central, by clicking **File** > **Open Interactive SQL**.
- From the **Start** menu by clicking **Start** > **Programs** > **SQL Anywhere 17** > **Administration Tools** > **Interactive SQL**.
- Using the `dbisql` command at a command prompt.

## Example

The following command runs the script file `mycom.sql` against the `CustDB.udb` database for UltraLite. Because a user ID and password are not defined, the default user ID `DBA` and password `sql` are assumed. The `-onerror` option is defined as `Exit`; so, if there is an error in the script file, the process terminates.

```
dbisql -ul -c DBF=CustDB.udb -onerror exit mycom.sql
```

## Related Information

[SQL Statements for Interactive SQL](#)

[Interactive SQL](#)

[SQL Statements for Interactive SQL](#)

[Interactive SQL](#)  
[Configuration Files](#)  
[Alphabetical List of Connection Parameters](#)  
[File Hiding Utility \(dbfhide\)](#)  
[INPUT Statement \[Interactive SQL\]](#)  
[OUTPUT Statement \[Interactive SQL\]](#)  
[READ Statement \[Interactive SQL\]](#)  
[UltraLite Connection Parameters \[page 181\]](#)  
[Supported Exit Codes \[page 214\]](#)  
[command\\_delimiter Option \[Interactive SQL\]](#)  
[default\\_isql\\_encoding Option \[Interactive SQL\]](#)  
[on\\_error Option \[Interactive SQL\]](#)

### 1.12.5.3 SQL preprocessor for UltraLite Utility (sqlpp)

Preprocesses a C/C++ program that contains Embedded SQL (ESQL), so that code required for that program can be generated before you run the compiler.

The table below describes the entire set of options for completeness, but the only relevant options for UltraLite are -eu and -wu.

#### ≡ Syntax

```
sqlpp -u [ options ] esql-filename [ output-filename ]
```

Option	Description
-d	Generate code that reduces data space size, but increases code size. Data structures are reused and initialized at execution time before use.

Option	Description
<code>-e flag</code>	<p>This option flags as an error any static Embedded SQL that is not part of a specified standard. The <code>level</code> value indicates the standard to use. For example, <code>sqlpp -e c03 . . .</code> flags any syntax that is not part of the core SQL/2003 standard.</p> <p>The allowed values of <code>level</code> are:</p> <ul style="list-style-type: none"> <li><b>c03</b> Flag syntax that is not core SQL/2003 syntax</li> <li><b>p03</b> Flag syntax that is not full SQL/2003 syntax</li> <li><b>c99</b> Flag syntax that is not core SQL/1999 syntax</li> <li><b>p99</b> Flag syntax that is not full SQL/1999 syntax</li> <li><b>e92</b> Flag syntax that is not entry-level SQL/1992 syntax</li> <li><b>i92</b> Flag syntax that is not intermediate-level SQL/1992 syntax</li> <li><b>f92</b> Flag syntax that is not full-SQL/1992 syntax</li> <li><b>t</b> Flag non-standard host variable types</li> <li><b>u</b> Flag syntax that is not supported by UltraLite</li> </ul> <p>For compatibility with previous SQL Anywhere versions, you can also specify <code>e</code>, <code>l</code>, and <code>f</code>, which correspond to <code>e92</code>, <code>i92</code>, and <code>f92</code>, respectively.</p>
<code>-h width</code>	<p>Limits the maximum length of split lines output by <code>sqlpp</code> to <code>width</code> in the <code>.c</code> file. Backslash characters are added to the end of split lines, so that a C compiler can parse the split lines as one continuous line. The default value is no maximum line length (output lines are not split by default).</p>
<code>-k</code>	<p>Notify the preprocessor that the program to be compiled includes a user declaration of <code>SQLCODE</code>.</p>
<code>-m mode</code>	<p>Cursor updatability mode. Either <i>HISTORICAL</i> or <i>READONLY</i>.</p>

Option	Description
<code>-n</code>	<p>Generate line number information in the C file by using <code>#line</code> directives in the appropriate places in the generated code.</p> <p>Use this option to the report source errors and to debug source on line numbers in the <code>esql-filename</code> file, rather than in the <code>output-filename</code> file.</p>
<code>-o O/S spec</code>	Not applicable to UltraLite.
<code>-q</code>	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
<code>-r-</code>	Not applicable to UltraLite.
<code>-S string-length</code>	Set the maximum size string that the preprocessor will put into the C file. Strings longer than this value are initialized using a list of characters ('a', 'b', 'c', and so on). Most C compilers have a limit on the size of string literal they can handle. This option is used to set that upper limit. The default value is 500.
<code>-u</code>	Required for UltraLite. Generate output specifically required for UltraLite databases.

Option	Description
-w <code>level</code>	<p>Flag non-conforming SQL syntax as a warning. The <code>level</code> value indicates the standard to use. For example, <code>sqlpp -w c03 . . .</code> flags any SQL syntax that is not part of the core SQL/2003 syntax.</p> <p>The allowed values of <code>level</code> are:</p> <p><b>c03</b> Flag syntax that is not core SQL/2003 syntax</p> <p><b>p03</b> Flag syntax that is not full SQL/2003 syntax</p> <p><b>c99</b> Flag syntax that is not core SQL/1999 syntax</p> <p><b>p99</b> Flag syntax that is not full SQL/1999 syntax</p> <p><b>e92</b> Flag syntax that is not entry-level SQL/1992 syntax</p> <p><b>i92</b> Flag syntax that is not intermediate-level SQL/1992 syntax</p> <p><b>f92</b> Flag syntax that is not full-SQL/1992 syntax</p> <p><b>t</b> Flag non-standard host variable types</p> <p><b>u</b> Flag syntax that is not supported by UltraLite</p> <p>For compatibility with previous SQL Anywhere versions, you can also specify <code>e</code>, <code>l</code>, and <code>f</code>, which correspond to <code>e92</code>, <code>i92</code>, and <code>f92</code>, respectively.</p>
-x	Change multibyte strings to escape sequences, so that they can be passed through a compiler.
-Z <code>collation-sequence</code>	Specify the collation sequence.

## Remarks

This preprocessor translates the SQL statements in the input-file into C/C++. It writes the result to the `output-filename`. The normal extension for source files containing Embedded SQL is `sql.c`. The default `output-filename` is the `esql-filename` base name with an extension of `c`. However, if the `esql-filename` already has the `.c` extension, the default output extension is `.cc`.

The collation sequence is used to help the preprocessor understand the characters used in the source code of the program. For example, in identifying alphabetic characters suitable for use in identifiers. In UltraLite, collations include a code page plus a sort order. If you do not specify `-z`, the preprocessor attempts to determine a reasonable collation to use based on the operating system.

To see a list of supported collations (and their corresponding codepages), run `ulinit -Z`.

### **i** Note

The SQL preprocessor (`sqlpp`) has the ability to flag static SQL statements in an Embedded SQL application at compile time. This feature can be especially useful when developing an UltraLite application, to verify SQL statements for UltraLite compatibility. You can test compatibility of SQL for both SQL Anywhere and UltraLite applications by using either `-e` and/or `-w` options.

## **Example**

The following command preprocesses the `srcfile.sql` Embedded SQL file in quiet mode for an UltraLite application.

```
sqlpp -u -q MyEsqFile.sql
```

## **Related Information**

[SQL Compliance Testing Using the SQL Flagger Embedded SQL](#)

[UltraLite Character Sets \[page 32\]](#)

### **1.12.5.4 UltraLite Engine Utility (uleng17)**

Manages concurrent UltraLite database connections from applications, and allows the UltraLite engine to run as a daemon using the `-ud` option.

#### **≡ Syntax**

```
uleng17 [ -ud ]
```

Option	Description
<code>-ud</code>	Lets you run the engine so that it continues running after the current user session ends. When you start the daemon directly using the <code>-ud</code> option, the <code>uleng17</code> command creates the daemon process and returns immediately (exiting and allowing the next command to be executed) before the daemon initializes itself or attempts to open any of the databases specified in the command.

## Remarks

The UltraLite engine does not display a messages window on startup.

The UltraLite engine should be used by an application in scenarios where multiple processes could be accessing the same database at the same time. The engine is installed in the SQL Anywhere `bin32` or `bin64` directory because the UltraLite desktop administration tools use the engine to connect to databases.

Using the `-ud` option, you can run the UltraLite engine so that when you log off the computer, the database engine remains running. (Normally when you log off the computer, all applications associated with the session shut down.)

## Related Information

[UltraLite Deployment \[page 123\]](#)

[How to Build and Deploy UltraLite C++ Applications \[page 667\]](#)

[UltraLite Data Management Components for Microsoft Windows Mobile \[page 21\]](#)

[UltraLite Engine Stop Utility \(ulstop\) \[page 224\]](#)

[UltraLite START Connection Parameter \[page 200\]](#)

### 1.12.5.5 UltraLite Engine Stop Utility (ulstop)

Stops the UltraLite engine.

☰, Syntax

`ulstop`



## Remarks

Use `ulstop` during development to shut down the engine manually. You typically do not require `ulstop` in live deployments.

## Related Information

[UltraLite Data Management Components for Microsoft Windows Mobile \[page 21\]](#)

[UltraLite Engine Utility \(`uleng17`\) \[page 223\]](#)

### 1.12.5.6 UltraLite Erase Utility (`ulerase`)

Erases an UltraLite database.

#### ≡ Syntax

```
ulerase [ options ] [ db-file-name ]
```

Option	Description
<code>@ data</code>	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.
<code>-k key</code> OR <code>--ek=key</code>	Specify the encryption key for an encrypted database.
<code>-p</code> OR <code>--ep</code>	Specify that you want to be prompted for the encryption key.
<code>--log</code>	Log operations to the specified file.
<code>-q</code> OR <code>--quiet</code>	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
<code>-u uid,pwd</code> OR <code>--dba=uid,pwd</code>	Specify the userid and password required to access the database.

Option	Description
-?	Displays utility usage information and exits.
OR	
--help	
db-file-name	Erase the specified database.

## Remarks

The database must be accessible. The user ID and password combination must allow a connection, otherwise the database is not erased.

Encrypted databases require a key provided in the connection string, or using one of *-k key* or *-p*.

## Related Information

[Configuration Files](#)

### 1.12.5.7 UltraLite Information Utility (ulinfo)

Displays information about an UltraLite database.

#### ≡ Syntax

```
ulinfo -c options
```

Option	Description
@ data	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.
-c "connection-string"	Supply database connection parameters. Required.
OR	
--connect="connection-string"	
-q	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
OR	
--quiet	

Option	Description
<code>--log=filename</code>	Log operations to the specified file.
<code>-?</code>	Displays utility usage information and exits.
OR	
<code>--help</code>	

## Remarks

Warning messages generated when opening an UltraLite database are always displayed unless you use the `-q` option.

## Example

Show basic database internals for a file named `cv_dbattr.udb` that has already been synchronized:

```
ulinfo -c DBF=cv_dbattr.udb
```

## Related Information

[Configuration Files](#)

[UltraLite Connection Parameters \[page 181\]](#)

[UltraLite global\\_database\\_id Option \[page 210\]](#)

[UltraLite ml\\_remote\\_id Option \[page 211\]](#)

### 1.12.5.8 UltraLite Initialize Database Utility (ulinit)

Creates a new UltraLite database.

This utility functions under one of the following modes:

#### Empty mode

Creates an empty database with characteristics specified with the command line arguments.

#### Extract mode

Creates a database based on a SQL Anywhere database.

An initial schema is created that matches tables and indexes in the SQL Anywhere reference database. Many of the reference database characteristics are extracted and used in the new UltraLite database.

## ☰ Syntax

```
ulinit options dbname
```

### i Note

If the mode is listed as Both In the table below, the option can be used in either empty or extract mode.

Option, Alternate Option	Description	Mode
@data	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.	Both
-a "keyword=value; ...", --SConnect="keyword=value; ..."	Sets the utility to <b>extract mode</b> and connects to an existing database using the specified connection parameters.  If this option is not present, the utility creates a new database using the specified connection parameters ( <b>empty mode</b> ).	Both
-c, --case	Enforce case sensitivity on all string comparisons.	Both
-d, --datacopy	For each table in the new UltraLite database, copy data from the corresponding table in the SQL Anywhere database. The new database is initially empty unless you use this option.  By default, this data is not uploaded in subsequent synchronizations. To include the data in the next upload synchronization, use -i with -d.	Extract
--date_format=format	Sets the format for dates retrieved from the database. Default is "YYYY-MM-DD".	Empty
--date_order=date-format-interpretation	Sets the interpretation of the date format. Default is "YMD".	Empty

Option, Alternate Option	Description	Mode
<code>-e value, --fips=value</code>	On or off, 1 or 0, and so on. This option controls the use of FIPS-certified AES encryption.	Both
<code>-f, --exactschema</code>	Fail if exact schema is not supported in UltraLite; otherwise, warnings will appear if schema differs.	Extract
<code>-g id, --databaseid=id</code>	Set the initial global database ID to the INTEGER value you assign. This initial value is used with a partition size for new rows that have global autoincrement columns. When deploying an application, you must assign a different range of identification numbers to each database for synchronization with the MobiLink server.	Both
<code>-i, --insertforupload</code>	Use with <code>-d</code> . Include inserted rows in the next upload synchronization. By default, rows inserted by this utility are not uploaded during synchronization.	Extract
<code>--identity-file=file</code>	Specify the file containing the client TLS, PEM, or PKCS12 identity.	Both
<code>--identity-password=password</code>	Specify the password for the client TLS identity.	Both
<code>-k key, --key=key</code>	Specify the encryption key for a new encrypted database.	Both
<code>-K, --prompt</code>	Specify that you want to be prompted for the encryption key.	Both
<code>-l filename, --sql=filename</code>	Log DDL database schema creation SQL statements, as executed, to the specified file.	Extract
<code>--log=filename</code>	Log operations to the specified file.	Both
<code>-m filename, --mirror_file=filename</code>	Specify the database mirror file.	Both
<code>--max_hash_size=size</code>	Sets the maximum default primary key or index hash size in bytes from 0 to 32. Default is 4.	Both

Option, Alternate Option	Description	Mode
<code>-n pubname, --publication=pubname</code>	<p>Required for extractions. Add tables to the UltraLite database schema.</p> <p><code>pubname</code> specifies a publication in the reference database. Tables in the publication are added to the UltraLite database.</p> <p>Specify the option multiple times to add tables from multiple publications to the UltraLite database. To add all tables in the reference database to the UltraLite database, specify <code>-n*</code>.</p>	Extract
<code>--nearest_century=yy</code>	Controls the interpretation of two-digit years in string-to-date conversions. Default is 50.	Empty
<code>-o value, --obfuscate=value</code>	On or off, 1 or 0, and so on. Controls whether data in the database is obfuscated. Obfuscation is not secure against skilled and determined attempts to gain access to the data. Default is 0.	Both
<code>-p size, --page_size=size</code>	Specify the database page size. Default is 4K.	Both
<code>--precision=precision</code>	Specifies the maximum number of digits in decimal point arithmetic results. Default is 30.	Empty
<code>-q, --quiet</code>	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages and warnings are still displayed, however.	Both
<code>-r size, -reserve_size</code>	Pre-allocate the file system space required for your UltraLite database, without actually inserting any data.	Both

Option, Alternate Option	Description	Mode
<code>-s pubname, --sync_publication</code>	<p>Create a publication in the UltraLite database with the same definition as <code>pubname</code> in the reference database. Publications are used to configure synchronization. Supply more than one <code>-s</code> option to name more than one synchronization publication.</p> <p>The tables in this publication must be included in a publication listed by the <code>-n</code> option.</p> <p>If <code>-s</code> is not supplied, the UltraLite remote database has no named publications.</p>	Extract
<code>-S checksum_level, --checksum_level=checksum_level</code>	0, 1, or 2. Specifies the checksum level validation on database pages. Default is 0 (checksum validation is disabled).	Both
<code>--scale=scale</code>	Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision. Default is 6.	Empty
<code>-t file, --rootcert=file</code>	Specify the file containing the trusted root certificate. This certificate is required for server authentication.	Both
<code>--time_format=format</code>	Sets the format for times retrieved from the database. Default is "HH:NN:SS.SSS".	Empty
<code>--timestamp_format=format</code>	Sets the format for timestamps retrieved from the database. Default is "YYYY-MM-DD HH:NN:SS.SSS".	Empty
<code>--timestamp_increment=increment</code>	Sets the resolution of timestamp values from 1 to 60000000 microseconds. Default is 1.	Empty
<code>--timestamp_with_time_zone_format=format</code>	This option sets the format for TIME- STAMP WITH TIME ZONE values retrieved from the database. Default is "YYYY-MM-DD HH:NN:SS.SSS +HH:NN".	Empty
<code>-u uid,pwd, --dba=uid,pwd</code>	<p>Database connection only.</p> <p>Specify the user ID and password.</p>	Both

Option, Alternate Option	Description	Mode
<code>--utf8_encoding=value</code>	On or off, 1 or 0, and so on. Encodes data using the UTF-8 character set format, 8-bit multibyte encoding for Unicode. Default is On.	Both
<code>-w, --nowarnings</code>	Do not display warnings.	Both
<code>-x table, --exclude</code>	Exclude the tables named in the list.	Extract
<code>-y, --overwrite</code>	Over-write the existing database file.	Both
<code>-z collation-sequence, --collation=collation-sequence</code>	Specify the collation sequence.	Empty
<code>-Z, --listcollation</code>	List the available collation sequences and exit.	Both
<code>-?, --help</code>	Display utility usage and exit.	Both

## Remarks

When run in extract mode, the `ulinit` utility attempts to create an UltraLite database that matches, as closely as possible, the SQL Anywhere database. For example, if a column in the SQL Anywhere database includes a clause that UltraLite does not support, the default value is ignored and the UltraLite default used instead. A warning is generated and creation continues. This supports the case where SQL Anywhere tables cannot be modified, but a reasonable UltraLite alternative is available. To enforce an exact schema match, use the `-f` option. The `ulinit` utility fails if the schema does not support a reasonable UltraLite alternative.

## Example

Create a file called `customer.udb` that contains the tables defined in `TestPublication`:

```
ulinit -a "DSN=MySADb;UID=JimmyB;PWD=secret" -n TestPublication -k mykey
customer.udb
```

This example connects to a SQL Anywhere database defined in the `MySADb` datasource. It creates an UltraLite database with all the database options from that database and all the tables contained in the `TestPublication` publication. The new UltraLite database is called `customer.udb` and is encrypted with the key `mykey`.



Create a file called `customer.udb` that contains two distinct publications. Specifically, `Pub1` may contain a small subset of data for priority synchronization, while `Pub2` could contain the bulk of the data:

```
ulinit -a "DSN=MySADb;UID=JimmyB;PWD=secret" --exactschema -n Pub1 -n Pub2 -s
Pub1 -s Pub2 customer.udb
```

This example connects to a SQL Anywhere database defined in the `MySADb` datasource. It creates an UltraLite database with all the database options from that database and all the tables contained in the publications `Pub1` and `Pub2`. The new UltraLite database is also created with the publications `Pub1` and `Pub2`. Since the `--exactschema` option is set, `ulinit` will fail if it cannot extract the all precise schema.

Create a new blank database that overwrites another `customer.udb` file if it already exists. The new database has no schema and all the database options are set to default values.

```
ulinit -y customer.udb
```

## Related Information

[Conversion from a SQL Anywhere Database to an UltraLite Database \[page 37\]](#)

[Synchronization Models](#)

[Configuration Files](#)

[Database Security \[page 35\]](#)

[Database Security \[page 35\]](#)

[UltraLite Character Sets \[page 32\]](#)

[Publishing Data in UltraLite \[page 83\]](#)

[UltraLite Connection Parameters \[page 181\]](#)

[UltraLite date\\_format Creation Option \[page 151\]](#)

[UltraLite date\\_order Creation Option \[page 153\]](#)

[UltraLite fips Creation Option \[page 159\]](#)

[UltraLite global\\_database\\_id Option \[page 210\]](#)

[identity MobiLink Client Network Protocol Option](#)

[identity\\_password MobiLink Client Network Protocol Option](#)

[UltraLite MIRROR\\_FILE Connection Parameter \[page 196\]](#)

[UltraLite max\\_hash\\_size Creation Option \[page 161\]](#)

[UltraLite nearest\\_century Creation Option \[page 162\]](#)

[UltraLite obfuscate Creation Option \[page 165\]](#)

[UltraLite Precision Creation Option \[page 168\]](#)

[UltraLite RESERVE\\_SIZE Connection Parameter \[page 199\]](#)

[UltraLite checksum\\_level Creation Option \[page 149\]](#)

[UltraLite scale Creation Option \[page 169\]](#)

[UltraLite time\\_format Creation Option \[page 171\]](#)

[UltraLite timestamp\\_format Creation Option \[page 173\]](#)

[UltraLite timestamp\\_increment Creation Option \[page 175\]](#)

[UltraLite utf8\\_encoding Creation Option \[page 180\]](#)

[UltraLite timestamp\\_with\\_time\\_zone\\_format Creation Option \[page 177\]](#)

## 1.12.5.9 UltraLite Load XML to Database Utility (ulload)

Loads data from an XML file into a new or existing database.

### ☰ Syntax

```
ulload -c "connection-string" [ options ] xml-file
```

Option	Description
@ data	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.
-a OR <i>--append</i>	Add data and schema definitions into an existing database.
-c "connection-string" OR <i>--connect="connection-string"</i>	Supply the database connection parameters.
-d OR <i>--dataonly</i>	Load data only, ignoring any schema metadata in the XML file input.  -d or --dataonly switches can only be used when -a is specified (because it is loading data only, the UDB it is loading the data into must exist with a schema that supports the data being loaded into it).
-e value OR <i>--fips= value</i>	Specify on or off, 1 or 0, and so on. This option controls the use of FIPS-certified AES encryption.

Option	Description
<p><code>-E behavior</code></p> <p>OR</p> <p><code>--onerror=behavior</code></p>	<p>Control what happens if an error is encountered while reading data from the XML file. Specify one of the following supported <code>behavior</code> values:</p> <p><b>continue</b></p> <p>uload ignores the error and continues to load XML.</p> <p><b>prompt</b></p> <p>uload prompts you to continue.</p> <p><b>quit</b></p> <p>uload stops loading the XML and terminates with an error. This behavior is the default behavior if none is specified.</p> <p><b>exit</b></p> <p>uload exits.</p>
<p><code>-f directory</code></p> <p>OR</p> <p><code>--filedir=directory</code></p>	<p>Set the directory that contains files with additional data to load.</p>
<p><code>-g ID</code></p> <p>OR</p> <p><code>--databaseid=ID</code></p>	<p>Set the initial global database ID to the INTEGER value you assign. This initial value is used with a partition size for new rows that have global autoincrement columns. When deploying an application, you must assign a different range of identification numbers to each database for synchronization with the MobiLink server.</p>
<p><code>-i</code></p> <p>OR</p> <p><code>--insertforsync</code></p>	<p>Include inserted rows in the next upload synchronization. By default, rows inserted by this utility are not uploaded during synchronization.</p>
<p><code>--identity-file = file</code></p>	<p>Specify the file containing the client TLS, PEM, or PKCS12 identity.</p>
<p><code>--identity-password = password</code></p>	<p>Specify the password for the client TLS identity.</p>
<p><code>-l filename</code></p> <p>OR</p> <p><code>--log=filename</code></p>	<p>Log operations to the specified file.</p>
<p><code>-n</code></p> <p>OR</p> <p><code>--schemaonly</code></p>	<p>Load schema metadata only, ignoring any data in the XML input file.</p>

Option	Description
-o <i>value</i> OR --obfuscate= <i>value</i>	On or off, 1 or 0, and so on. Controls whether data in the database is obfuscated. Obfuscation is not secure against skilled and determined attempts to gain access to the data.
-p <i>page-size</i> OR --page_size= <i>page-size</i>	Defines the database page size.
-q OR --quiet	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
-s <i>file</i> OR --sql= <i>file</i>	Log the SQL statements used to load the database into the specified <i>file</i> .
-t <i>file</i> OR --rootcert= <i>file</i>	Specify the file containing the trusted root certificate. This certificate is required for server authentication.
--utf8_encoding= <i>value</i>	On or off, 1 or 0, and so on. Encodes data using the UTF-8 character set format, 8-bit multibyte encoding for Unicode.
-v OR --verbose	Print verbose messages.
-y OR --overwrite	Overwrite the database file without confirmation. This only applies when you use ulload to create a new database.
-? OR --help	Display the utility usage and exit.

## Remarks

The ulload utility takes an input XML file generated by ulunload, ulunloadold (provided with SQL Anywhere 10), or ulxml (in UltraLite versions 8 and 9). When used along with ulunload this utility provides you with the ability

to rebuild a database. An alternative method to rebuild a database is using ulunload to generate SQL statements and then use DBISQL to read them into a new database.

The XML file can contain metadata for the schema and/or metadata for the database data. -d ignores the schema metadata, only adding data to the .udb file. -n ignores the data and the metadata, only adding the schema to the .udb file.

Setting an option or specifying a certificate on the command line overrides any settings in the `xml-file` that is processed by ulload.

The ulload utility restores any synchronization profiles to the database when reading the XML.

This utility returns error codes. Any value other than 0 means that the operation failed.

## Example

Create a new UltraLite database file, `sample.udb`, and load it with data in `sample.xml`:

```
ulload -c DBF=sample.udb sample.xml
```

Load the data from `sample.xml` into the existing database `sample.udb`, and if an error occurs, prompt for action:

```
ulload -d -c DBF=sample.udb --onerror=prompt sample.xml
```

Create the schema and data stored in `test_data.xml` in the `sample.udb` database. Since the -a switch is specified, `sample.udb` must exist prior to running this command. Moreover, any schema or data that conflicts with what is already in `sample.udb` will mean the ULLOAD command will fail.

```
ulload -c DBF=sample.udb -a test_data.xml
```

## Related Information

[Configuration Files](#)

[Database Security \[page 35\]](#)

[Database Security \[page 35\]](#)

[UltraLite Character Sets \[page 32\]](#)

[UltraLite Connection Parameters \[page 181\]](#)

[UltraLite Database Unload Utility \(ulunload\) \[page 243\]](#)

[Supported Exit Codes \[page 214\]](#)

[UltraLite fips Creation Option \[page 159\]](#)

[UltraLite Database Unload Utility \(ulunload\) \[page 243\]](#)

[UltraLite global\\_database\\_id Option \[page 210\]](#)

[identity MobiLink Client Network Protocol Option](#)

[identity\\_password MobiLink Client Network Protocol Option](#)

[UltraLite obfuscate Creation Option \[page 165\]](#)

[UltraLite page\\_size Creation Option \[page 166\]](#)

[UltraLite global\\_database\\_id Option \[page 210\]](#)

[UltraLite utf8\\_encoding Creation Option \[page 180\]](#)

## 1.12.5.10 UltraLite Synchronization Utility (ulsync)

Synchronizes an UltraLite database with a MobiLink server. This tool can be used for testing synchronization during application development.

### ☰ Syntax

```
ulsync [ options ] [synchronization-parameters]
```

Option	Description
@ data	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.
-c "connection-string" or --connect="connection-string"	Required. Connect to the database as identified in the DBF or file_name parameter of your connection-string. If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-p profile-name or --profile=profile	Synchronize using the named synchronization profile, equivalent to: <pre>SYNCHRONIZE profileName MERGE syncOptions</pre> where sync options are taken from the trailing ulsync options. For example: <pre>ulsync -p profileName "MobiLinkUid=ml;ScriptVersion=Version 001...syncOptions"</pre>
-q or --quiet	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
-r or --result	Display last synchronization results and exit.

Option	Description
-v or <code>--verbose</code>	Display synchronization progress messages. This also determines whether progress is displayed for any synchronization, whether using the C++ API or the SQL SYNCHRONIZE PROFILE statement.
<code>--log=filename</code>	Log operations to the specified file.
-? or <code>--help</code>	Display utility usage information and exit.

## Remarks

Your certificate may be bundled with your application by including it in the Xcode project. If you specify the bare name plus extension for a certificate file option, UltraLite will automatically look in the bundle to find it. If you want, you can specify a full path.

The following options that were valid for versions 10 and earlier are no longer supported: -a `authenticate-parameters`, -e `sync-parms`, -k `stream-type`, -n (no sync), and -x `protocol options`. -e `keyword=value` is now part of the sync parameters string and -k and -x are now part of the `Stream= stream{stream-parms}` sync parameters string.

Below, we show a `ulsync` example and the equivalent SQL statement.

```
ulsync -p profile "parms"
```

This command is equivalent to the following SQL statement.

```
SYNCHRONIZE PROFILE profile MERGE parms
```

The following is another example.

```
ulsync "parms"
```

This command is equivalent to the following SQL statement.

```
SYNCHRONIZE USING parms
```

For secure synchronization, the UltraLite application must have access to the public certificate. You can reference a certificate by:

- Incorporating the certificate information into the UltraLite database at creation time with the `-t file` option using `ulinit` or `ulload`.
- Referencing an external certificate file at synchronization time with the `trusted_certificate= file` stream option.

This utility returns error codes. Any value other than 0 means that the operation failed.

## Example

The following command synchronizes a database file called `myuldb.udb` for a MobiLink user called `remoteA`.

```
ulsync -c DBF=myuldb.udb "MobiLinkUid=remoteA;Stream=http;ScriptVersion=2"
```

The following command synchronizes a database file called `myuldb.udb` over HTTPS with the `C:\Users\Public\Documents\SQL Anywhere 17\Samples\Certificates\rsaroot.crt` certificate. The `trusted_certificate=file` option must be used because the trusted certificate file was not added to the database when the database was created. Additionally, the MobiLink user name is `remoteB`.

```
ulsync -c DBF=myuldb.udb "Stream=https{trusted_certificate=C:\Users\Public\Documents\SQL Anywhere 17\Samples\Certificates\rsaroot.crt};MobiLinkUid=remoteB;ScriptVersion=2;UploadOnly=ON"
```

The following command displays the last synchronization results for a database file named `synced.udb`.

```
ulsync -r -c dbf=synced.udb
```

The previous synchronization results are listed as follows:

```
SQL Anywhere UltraLite Synchronization Utility Version 17.0.11.1293
Results of last synchronization:
Succeeded
  Download timestamp: 2006-07-25 16:39:36.708000
  Upload OK
  No ignored rows
  Partial download retained
  Authentication value: 1000 (0x3e8)
```

The following example shows the command line used to synchronize the `CustDB` database with a user name of `50` over TCP/IP on a port of `2439`. It uses verbose progress messages.

```
ulsync -c "dbf=C:\Users\Public\Documents\SQL Anywhere 17\Samples\UltraLite\CustDB\custdb.udb"
"MobiLinkUid=50;ScriptVersion=custdb 17.0.11;Stream=tcPIP{port=2439}"
```

The following command illustrates how to use TLS encryption with end-to-end encryption (E2EE):

```
ulsync -c "uid=dba;pwd=sql;dbf=myuldb.db"
"MobiLinkUid=rem1;MobiLinkPwd=password;ScriptVersion=v1;Stream=tls{host=myServer;port=2439;trusted_certificate=clientcert.pem;e2ee_public_key=e2eepublic.pem}"
```

## Related Information

[End-to-end Encryption](#)

[UltraLite Clients \[page 73\]](#)

[Configuration Files](#)

[UltraLite Synchronization Profile Options \[page 241\]](#)

[trusted\\_certificates MobiLink Client Network Protocol Option](#)

[UltraLite Connection Parameters \[page 181\]](#)



[Supported Exit Codes \[page 214\]](#)

[MobiLink File Transfer Utility \(mfiletransfer\)](#)

[UltraLite Synchronization Profile Options \[page 241\]](#)

[CREATE SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 535\]](#)

[trusted\\_certificates MobiLink Client Network Protocol Option](#)

## 1.12.5.11 UltraLite Synchronization Profile Options

Specify synchronization profile options with the `ulsync` utility on the command line after you have defined all other command line options. Keywords are case insensitive.

Synchronization profile option	Valid values	Description
<code>AllowDownloadDupRows</code>	Boolean	This option prevents errors from being raised when multiple rows are downloaded that have the same primary key. This can be used to allow inconsistent data to be synchronized without causing the synchronization to fail. The default value is "no."
<code>AuthParms</code>	String (comma separated)	Specifies the list of authentication parameters sent to the MobiLink server. You can use authentication parameters to perform custom authentication in MobiLink scripts.
<code>CheckpointStore</code>	Boolean	Adds additional checkpoints of the database during synchronization to limit database growth during the synchronization process.
<code>ContinueDownload</code>	Boolean	Restarts a previously failed download. When continuing a download, only the changes that were selected to be downloaded with the failed synchronization are received. By default, UltraLite does not continue downloads.
<code>DisableConcurrency</code>	Boolean	Disallow database access from other threads during synchronization.
<code>DownloadOnly</code>	Boolean	Performs a download-only synchronization.
<code>KeepPartialDownload</code>	Boolean	Controls whether UltraLite keeps a partial download if a communication error occurs. By default, UltraLite does not roll back partially downloaded changes.
<code>MobiLinkPwd</code>	String	Specifies the existing MobiLink password associated with the user name.
<code>MobiLinkUid</code>	String	Specifies the MobiLink user name.

Synchronization profile option	Valid values	Description
NewMobiLinkPwd	String	Supplies a new password for the MobiLink user. Use this option when you want to change an existing password.
Ping	Boolean	Confirms communications with the server only; no synchronization is performed.
Publications	String (comma separated)	Specifies the publications(s) to synchronize. The publications determine the tables on the remote database that are involved in synchronization. If this parameter is blank (the default) then all tables are synchronized. If the parameter is an asterisk (*) then all publications are synchronized.
ScriptVersion	String	Specifies the MobiLink script version. The script version determines which scripts are run by MobiLink on the consolidated database during synchronization. If you do not specify a script version, 'default' is used.
SendDownloadACK	Boolean	Specifies that a download acknowledgement should be sent from the client to the server. By default, the MobiLink server does not provide a download acknowledgement.
Stream	String (with sub-list)	Specifies the MobiLink network synchronization protocol.
TableOrder	String (comma separated)	Specifies the order of tables in the upload. By default, UltraLite selects an order based on foreign key relationships.
UploadOnly	String	Specifies that synchronization will only include an upload, and no download will occur.

The Boolean values can be specified as Yes/No, 1/0, True/False, On/Off. In all the Boolean cases, the default is No. For all other values, the default is simply unspecified.

## Related Information

[Resumption of Failed Downloads](#)

[Publishing Data in UltraLite \[page 83\]](#)

[Additional Parameters Synchronization Parameter \[page 95\]](#)

[Authentication Parameters Synchronization Parameter \[page 98\]](#)

[Download Only Synchronization Parameter \[page 101\]](#)

[Keep Partial Download Synchronization Parameter \[page 103\]](#)

[MobiLinkPwd \(mp\) Extended Option](#)

- [-mn dbmlsync Option](#)
- [Ping Synchronization Parameter \[page 108\]](#)
- [ScriptVersion \(sv\) Extended Option](#)
- [Send Download Acknowledgement Synchronization Parameter \[page 111\]](#)
- [Stream Type Synchronization Parameter \[page 114\]](#)
- [Upload Only Synchronization Parameter \[page 118\]](#)
- [ALTER SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 522\]](#)
- [DROP SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 547\]](#)
- [SYNCHRONIZE Statement \[UltraLite\] \[page 563\]](#)
- [UltraLite Options \[page 143\]](#)

## 1.12.5.12 UltraLite Database Unload Utility (ulunload)

Unloads either an entire UltraLite database to XML or SQL, or all or part of UltraLite data to XML or SQL.

### ≡ Syntax

```
ulunload -c "connection-string" [ options ] output-file
```

Option	Description
@ data	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.
-b max-size OR --maxblob=max-size	Set the maximum size of column data to be stored in the XML file. The default is 10 KB. To store all data in the XML file (no maximum size), use -b -1.
-c "connection-string" OR --connect="connection-string"	Required. Connect to the database as identified in the DBF or file_name parameter of your connection-string. If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-d OR --dataonly	Only unload the data from the database to the output file. Do not unload any schema information.
-e table,... OR --exclude=table,...	Exclude the named table when unloading the database. You can name multiple tables in a comma-separated list. For example: <pre>-e mydbtable1,mydbtable5</pre>

Option	Description
<code>-f directory</code> OR <code>--filedir=directory</code>	Set the directory to store data larger than the maximum size specified by <code>-b</code> . The default is the same directory as the output file.
<code>-l filename</code> OR <code>--log=filename</code>	Log operations to the specified file.
<code>-n</code> OR <code>--schemaonly</code>	Unload schema only, ignoring any data in the database.
<code>-q</code> OR <code>--quiet</code>	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
<code>-s</code> OR <code>--sql</code>	Unload as SQL Anywhere-compatible SQL statements. SQL file output can be read by UltraLite or SQL Anywhere using DBISQL.
<code>-t table...</code> OR <code>--include=table...</code>	Unload data in the named <code>table</code> only. You can name multiple tables in a comma separated list. For example: <pre>-t mydbtable2,mydbtable6</pre>
<code>-v</code> OR <code>--verbose</code>	Print verbose messages.
<code>-x owner</code> OR <code>--owner=owner</code>	Output tables so they are owned by a specific user ID. You can use this option with the <code>-s</code> option.
<code>-y</code> OR <code>--overwrite</code>	Overwrite <code>output-file</code> without confirmation.
<code>-?</code> OR <code>--help</code>	Displays utility usage information and exits.

Option	Description
<code>output-file</code>	Required. Set the name of the file that the database is unloaded into. If you use the <code>-s</code> option, database is unloaded as SQL statements. Otherwise, the database is unloaded as XML.

## Remarks

By default, `ulunload` outputs XML that describes the schema and data in the database. You can use the output for archival purposes, or to keep the UltraLite database portable across all releases.

Saving a database with a synchronization profile results in XML that is incompatible with earlier versions of the UltraLite utilities. A workaround is to edit the XML and remove the text section marked with

```
<syncprofiles>...</syncprofiles>
```

Unloading a database does not preserve:

- Synchronization state, stored synchronization counts, and row deletions. Ensure you synchronize the database before unloading it.
- UltraLite user entries.

To confirm what database options or properties have been preserved, run `ulinfo` after you have reloaded your database with the `ulload` utility.

If column data exceeds the maximum size you specified with `-b`, the overflow is saved to a `*.bin` file in either:

- the same directory as the XML file
- the directory specified by `-f`.

The file follows this naming convention:

```
tablename-columnname-rownumber.bin
```

The `-x` option allows you to assign ownership to UltraLite tables. You only need to assign an owner to a table if you intend to use the resulting SQL statements for creating or modifying a SQL Anywhere database. When read by UltraLite, the owner names are silently ignored.

This utility returns error codes. Any value other than 0 means that the operation failed.

If you are using this utility to unload a database on the Microsoft Windows Mobile device directly, UltraLite cannot back up the database before the unload or action occurs. You must perform this action manually before running these wizards.

## Example

Unload the `sample.udb` database into the `sample.xml` file.

```
ulunload -c DBF=sample.udb sample.xml
```

Unload the data from the `sample.udb` database into a SQL file called `sample.sql`. Overwrite the SQL file if it exists.

```
ulunload -c DBF=sample.udb -d -y -s sample.sql
```

## Related Information

[Configuration Files](#)

[UltraLite Connection Parameters \[page 181\]](#)

[Supported Exit Codes \[page 214\]](#)

[UltraLite Load XML to Database Utility \(ulload\) \[page 234\]](#)

[UltraLite Information Utility \(ulinfo\) \[page 226\]](#)

### 1.12.5.13 UltraLite Validate Database Utility (ulvalid)

Performs a full (normal) validation of an UltraLite database.

#### ☰ Syntax

```
ulvalid -c "connection-string" [ options ]
```

Option	Description
@ data	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.
-c "connection-string" OR --connect="connection-string"	Required. Connect to the database as identified in <code>connection-string</code> . If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-e OR --express	Express validation. Only perform table validation. This option provides a faster validation than normal validation.

Option	Description
-q OR <i>--quiet</i>	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
-v OR <i>--verbose</i>	Print verbose messages.
<i>--log=filename</i>	Log operations to the specified file.
-? OR <i>--help</i>	Displays utility usage information and exits.

## Remarks

Validating a database verifies the accuracy of the table metadata and ensures the file has not been corrupted.

The validation includes:

### Database pages

Validate all database pages, using checksums when enabled. Certain critical pages always have checksums and even pages without checksums undergo a basic validity check.

### Tables

Validate table(s) by checking that the table row count matches the count in each index.

### Indexes

Validate indexes by checking that entries refer to valid rows. `ulvalid -e` performs an express check, which includes only table validation.

## Example

An example of an express validation of a database named `sample.udb` run in quiet mode.

```
ulvalid -c DBF=sample.udb -e -q
```

## Related Information

[Configuration Files](#)

[Validating an UltraLite Database \[page 51\]](#)

[UltraLite checksum\\_level Creation Option \[page 149\]](#)

### 1.12.6 UltraLite System Tables

The schema of an UltraLite database is stored in a proprietary format.

Earlier versions of UltraLite databases were stored in several system tables. These system tables (system views) can still be queried for backward compatibility, but they only contain information about user schema (like tables, columns, indexes) not system schema. For example, you cannot query the systable to find the properties of systable itself. You can only query the systable to find the properties of user-created tables.

Each UltraLite programming API supports objects and methods that can be used to query the database about its schema. Use these objects and APIs to explore schema rather than querying the system views.

All queries performed on these system views are equivalent to full table scans. Index scans are not supported on these system views.

#### In this section:

[sysarticle System Table \[page 249\]](#)

Each row in the sysarticle system table describes a table that belongs to a publication.

[syscolumn System Table \[page 249\]](#)

Each row in the syscolumn system table describes one column.

[sysindex System Table \[page 250\]](#)

Each row in the sysindex system table describes one index in the database.

[sysixcol System Table \[page 251\]](#)

Each row in the sysixcol system table describes one column of an index listed in sysindex.

[syspublication System Table \[page 252\]](#)

Each row in the syspublication system table describes a publication.

[syssynresult System Table \[page 252\]](#)

Any row in the syssynresult system table contains information about the most recent synchronization.

[systable System Table \[page 254\]](#)

Each row in the systable system table describes one table in the database.



## 1.12.6.1 sysarticle System Table

Each row in the sysarticle system table describes a table that belongs to a publication.

Column name	Column type	Description
publication_id	UNSIGNED INT	An identifier for the publication that this article belongs to.
table_id	UNSIGNED INT	The identifier of the table that belongs to the publication.
where_expr	VARCHAR(2048)	An optional predicate to filter rows.

### Constraints

PRIMARY KEY (publication\_id, table\_id)

FOREIGN KEY (publication\_id) REFERENCES syspublication (publication\_id)

FOREIGN KEY (table\_id) REFERENCES systable (object\_id)

### Related Information

[syspublication System Table \[page 252\]](#)

## 1.12.6.2 syscolumn System Table

Each row in the syscolumn system table describes one column.

Column name	Column type	Description
column_name	VARCHAR(128)	The name of the column.
default	VARCHAR(128)	The default value for this column. For example, autoincrement.
domain	UNSIGNED INT	The column domain, which is an enumerated value indicating the domain of the column.
domain_info	UNSIGNED INT	Used with a variable sized domain.
nulls	VARCHAR(1)	Determines if the column allows nulls default.
object_id	UNSIGNED INT	A unique identifier for that column.

Column name	Column type	Description
table_id	UNSIGNED INT	The identifier of the table to which the column belongs.

## Constraints

PRIMARY KEY( table\_id, object\_id )

FOREIGN KEY (table\_id) REFERENCES systable (object\_id)

### 1.12.6.3 sysindex System Table

Each row in the sysindex system table describes one index in the database.

Column name	Column type	Description
check_on_commit	BIT	Indicates when referential integrity is checked to ensure there is a matching primary row for every foreign key. It is only required if type is <i>foreign</i> .
index_name	VARCHAR(128)	The name of the index.
ixcol_count	UNSIGNED INT	The number of columns in the index.
nullable	BIT	Only required if type is <i>foreign</i> . Indicates if nulls are allowed.
object_id	UNSIGNED INT	A unique identifier for an index.
primary_index_id	UNSIGNED INT	Only required if type is <i>foreign</i> . Lists the identifier of the primary index.
primary_table_id	UNSIGNED INT	Only required if type is <i>foreign</i> . Lists the identifier of the primary table.
root_handle	UNSIGNED INT	For internal use only.
table_id	UNSIGNED INT	A unique identifier for the table to which the index applies.
type	VARCHAR(10)	The type of index. Can be one of: <ul style="list-style-type: none"> <li>• <i>primary</i></li> <li>• <i>foreign</i></li> <li>• <i>key</i></li> <li>• <i>unique</i></li> <li>• <i>index</i></li> </ul>
hash_size	UNSIGNED SHORTINT	Stores the hash size used for index hashing.

## Constraints

PRIMARY KEY( table\_id, object\_id)

FOREIGN KEY( table\_id ) REFERENCES systable( object\_id )

## Related Information

[sysixcol System Table \[page 251\]](#)

### 1.12.6.4 sysixcol System Table

Each row in the sysixcol system table describes one column of an index listed in sysindex.

Column name	Column type	Description
column_id	UNSIGNED INT	A unique identifier for the column being indexed.
index_id	UNSIGNED INT	A unique identifier for the index that this index-column belongs to.
order	VARCHAR(1)	Indicates whether the column in the index is kept in ascending (A) or descending (D) order.
sequence	UNSIGNED INT	The order of the column in the index.
table_id	UNSIGNED INT	A unique identifier for the table to which the index applies.

## Constraints

PRIMARY KEY( table\_id, index\_id, sequence )

FOREIGN KEY( table\_id, index\_id ) REFERENCES sysindex( table\_id, object\_id )

FOREIGN KEY( table\_id, column\_id ) REFERENCES syscolumn( table\_id, object\_id )

## Related Information

[sysindex System Table \[page 250\]](#)

## 1.12.6.5 syspublication System Table

Each row in the syspublication system table describes a publication.

Column name	Column type	Description
download_timestamp	TIMESTAMP	The time of the last download.
last_sync	UNSIGNED BIGINT	Used to keep track of upload progress.
publication_id	UNSIGNED INT	A unique identifier for the publication.
publication_name	VARCHAR(128)	The name of the publication.

### Constraints

PRIMARY KEY (publication\_id)

### Related Information

[sysarticle System Table \[page 249\]](#)

## 1.12.6.6 syssynctest System Table

Any row in the syssynctest system table contains information about the most recent synchronization.

Column name	Column type	Description
sql_code	INTEGER	The SQL code from the last synchronization.
error_string	CHAR(200)	The error message from the last synchronization.
stream_error_code	SMALLINT	The specific stream error. See the ss_error_code enumeration for possible values.
system_error_code	INTEGER	A system-specific error code. For more information about error codes, see your platform documentation.
stream_error_string	CHAR(80)	A string with additional information, if available, for the stream_error_code value.
upload_ok	BIT	True if the upload was successful; false otherwise.

Column name	Column type	Description
ignored_rows	BIT	True if uploaded rows were ignored; false otherwise.
auth_status	UNSIGNED SMALLINT	The synchronization authentication status.
auth_value	INTEGER	The value used by the MobiLink server to determine the auth_status result.
auth_info	CHAR(1024)	The authentication message returned from the MobiLink user authentication script.
partial_download_retained	BIT	The value that tells you whether a partial download was retained.
timestamp	TIMESTAMP	The time and date of the last synchronization.
sent_bytes	UNSIGNED INT	The number of bytes currently sent for the upload.
sent_inserts	UNSIGNED INT	The number of rows currently inserted for the upload.
sent_updates	UNSIGNED INT	The number of updated rows currently sent for the upload.
sent_deletes	UNSIGNED INT	The number of deleted rows currently sent for the upload.
received_bytes	UNSIGNED INT	The number of bytes currently sent for the download.
received_inserts	UNSIGNED INT	The number of rows currently inserted for the download.
received_updates	UNSIGNED INT	The number of updated rows currently sent for the download.
received_ignored_updates	UNSIGNED INT	The number of duplicate rows that were received in the download.
received_deletes	UNSIGNED INT	The number of deleted rows currently sent for the download.
received_ignored_deletes	UNSIGNED INT	The number of deleted rows that were received in the download of rows that have already been deleted.
received_truncate_deletes	UNSIGNED INT	The number of rows that were deleted in the download by a truncate operation.

## Related Information

[Keep Partial Download Synchronization Parameter \[page 103\]](#)

## 1.12.6.7 systable System Table

Each row in the systable system table describes one table in the database.

Column name	Column type	Description
column_count	UNSIGNED INT	The number of columns in the table.
index_count	UNSIGNED INT	The number of indexes in the table.
ixcol_count	UNSIGNED INT	The total number of columns in all indexes in the table.
table_name	VARCHAR(128)	The name of the table.
object_id	UNSIGNED INT	A unique identifier for that table.
sync_type	VARCHAR(32)	Used for MobilLink synchronization. Can be one of either <i>no_sync</i> for no synchronization, <i>all_sync</i> to synchronize every row, or <i>normal_sync</i> for synchronize changed rows only.
table_type	VARCHAR(32)	<i>user</i> to indicate user-created tables.

### Constraints

PRIMARY KEY (object\_id)

## 1.13 UltraLite SQL reference

UltraLite supports many SQL language features and elements.

### In this section:

[UltraLite SQL Language Elements \[page 255\]](#)

UltraLite supports many SQL elements.

[SQL Data Types \[page 288\]](#)

UltraLite supports a subset of the data types available in SQL Anywhere.

[Spatial Data Types \[page 320\]](#)

**Spatial data** is data that describes the position, shape, and orientation of objects in a defined space.

UltraLite provides storage and data management features for spatial data, in the form of points, allowing you to store information such as geographic locations and routing information.

[User-defined Data Types and Their Equivalents \[page 322\]](#)

Unlike SQL Anywhere databases, UltraLite does not support user-defined data types.

[SQL Functions \[page 323\]](#)

Functions are used to return information from the database. They can be called anywhere an expression is allowed.

[UltraLite SQL Statements \[page 516\]](#)

The SQL statements supported by UltraLite SQL are a subset of the statements supported by SQL Anywhere databases.

## 1.13.1 UltraLite SQL Language Elements

UltraLite supports many SQL elements.

### In this section:

[Keywords in UltraLite \[page 256\]](#)

Each SQL statement contains one or more keywords. SQL keywords are case insensitive, but throughout the documentation, keywords are indicated in uppercase. Some keywords cannot be used as identifiers without surrounding them in double quotes. These are called **reserved words**.

[Identifiers in UltraLite \[page 256\]](#)

Identifiers are the names of objects in the database, such as user IDs, tables, and columns.

[Strings in UltraLite \[page 257\]](#)

Strings are used to hold character data in the database. UltraLite supports the same rules for strings as SQL Anywhere.

[Comments in UltraLite \[page 257\]](#)

Comments are used to attach explanatory text to SQL statements or statement blocks. The UltraLite runtime does not execute comments.

[Numbers in UltraLite \[page 258\]](#)

Numbers are used to hold numerical data in the database.

[The NULL Value in UltraLite \[page 259\]](#)

As with SQL Anywhere, NULL is a special value that is different from any valid value for any data type. However, the NULL value is a legal value in any data type. NULL is used to represent unknown (no value) or inapplicable information.

[Special Values in UltraLite \[page 259\]](#)

You can use special values in expressions, and as column defaults when you create tables.

[Dates and Times in UltraLite \[page 264\]](#)

Many of the date and time functions use dates built from date and time parts. UltraLite and SQL Anywhere support the same date parts.

[Expressions in UltraLite \[page 264\]](#)

Expressions are formed by combining data, often in the form of column references, with operators or functions.

[Search Conditions in UltraLite \[page 272\]](#)

A search condition is the criteria for a WHERE clause, a HAVING clause, an ON phrase in a join, or an IF expression. A search condition is also called a **predicate**.

[Operators in UltraLite \[page 285\]](#)

Operators are used to compute values, which may in turn be used as operands in a higher-level expression.

[Variables in UltraLite \[page 288\]](#)

You cannot use SQL variables (including global variables) in UltraLite applications.

## 1.13.1.1 Keywords in UltraLite

Each SQL statement contains one or more keywords. SQL keywords are case insensitive, but throughout the documentation, keywords are indicated in uppercase. Some keywords cannot be used as identifiers without surrounding them in double quotes. These are called **reserved words**.

### i Note

UltraLite only supports a subset of SQL Anywhere keywords. However, to avoid potential problems in future releases, you should assume that all the reserved words for SQL Anywhere apply to UltraLite as well.

## Related Information

[Reserved Words](#)

## 1.13.1.2 Identifiers in UltraLite

Identifiers are the names of objects in the database, such as user IDs, tables, and columns.

Identifiers have a maximum length of 128 bytes and are composed from alphabetic characters and digits, as well as the underscore character (\_) and at sign (@). Leading digits are allowed but the identifier must be quoted. Other special characters are allowed but the identifier must be quoted. The database collation sequence dictates which characters are considered alphabetic or digit characters.

Double quotes (") are not permitted in identifiers:

## Quoting Identifiers

If any of the following conditions are true, then always enclose an identifier in double quotes:

- The identifier contains leading, trailing, or embedded spaces.
- The first character of the identifier is not an alphabetic character, the underscore character (\_), or at sign (@). For example, the first character is a digit.
- The identifier contains characters other than the alphabetic characters, digits, underscore character (\_), and at sign (@).
- The identifier is a reserved word.

For compatibility with other database management systems, it is recommended that you avoid the use of special characters in identifier names, including but not limited to any of the following:

- Leading or trailing whitespace
- Leading single quote
- Semicolons



## Related Information

[Reserved Words](#)

### 1.13.1.3 Strings in UltraLite

Strings are used to hold character data in the database. UltraLite supports the same rules for strings as SQL Anywhere.

The results of comparisons on strings, and the sort order of strings, depends on the case sensitivity of the database, the character set, and the collation sequence. These properties are set when the database is created.

## Related Information

[Strings](#)

[UltraLite Character Sets \[page 32\]](#)

### 1.13.1.4 Comments in UltraLite

Comments are used to attach explanatory text to SQL statements or statement blocks. The UltraLite runtime does not execute comments.

The following comment indicators are available in UltraLite:

#### -- (Double hyphen)

The database server ignores any remaining characters on the line. This indicator is the ANSI/ISO SQL Standard comment indicator.

#### // (Double slash)

The double slash has the same meaning as the double hyphen.

#### /\* ... \*/ (Slash-asterisk)

Any characters between the two comment markers are ignored. The two comment markers may be on the same or different lines. Comments indicated in this style can be nested. This style of commenting is also called C-style comments.

#### i Note

The percent sign (%) is not supported in UltraLite.

## Example

- The following example illustrates the use of double-hyphen comments:

```
CREATE TABLE borrowed_book (
  loaner_name CHAR(100) PRIMARY KEY,
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
  date_returned DATE,
  book CHAR(20)
  FOREIGN KEY book REFERENCES library_books (isbn),
)
--This statement creates a table for a library database to hold information
on borrowed books.
--The default value for date_borrowed indicates that the book is borrowed on
the day the entry is made.
--The date_returned column is NULL until the book is returned.
```

- The following example illustrates the use of C-style comments:

```
CREATE TABLE borrowed_book (
  loaner_name CHAR(100) PRIMARY KEY,
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
  date_returned DATE,
  book CHAR(20)
  FOREIGN KEY book REFERENCES library_books (isbn),
)
/* This statement creates a table for a library database to hold information
on borrowed books.
The default value for date_borrowed indicates that the book is borrowed on
the day the entry is made.
The date_returned column is NULL until the book is returned. */
```

### 1.13.1.5 Numbers in UltraLite

Numbers are used to hold numerical data in the database.

A number can:

- be any sequence of digits
- be appended with decimal parts
- include an optional negative sign (-) or a plus sign (+)
- be followed by an e and then a numerical exponent value

For example, all numbers shown below are supported by UltraLite:

42

-4.038

.001

3.4e10

1e-10

## 1.13.1.6 The NULL Value in UltraLite

As with SQL Anywhere, NULL is a special value that is different from any valid value for any data type. However, the NULL value is a legal value in any data type. NULL is used to represent unknown (no value) or inapplicable information.

### Related Information

[NULL Special Value](#)

## 1.13.1.7 Special Values in UltraLite

You can use special values in expressions, and as column defaults when you create tables.

### In this section:

[CURRENT DATE Special Value - UltraLite \[page 260\]](#)

Returns the current year, month, and day.

[CURRENT TIME Special Value - UltraLite \[page 260\]](#)

The current hour, minute, second, and fraction of a second.

[CURRENT TIMESTAMP Special Value - UltraLite \[page 261\]](#)

Combines CURRENT DATE and CURRENT TIME to form a TIMESTAMP value containing the year, month, day, hour, minute, second, and fraction of a second.

[CURRENT UTC TIMESTAMP Special Value - UltraLite \[page 262\]](#)

Returns a TIMESTAMP WITH TIME ZONE value that reflects the current UTC time containing the year, month, and day.

[SQLCODE Special Value \[page 263\]](#)

Current SQLCODE value at the time the special value was evaluated.

### Related Information

[SQL Data Types \[page 288\]](#)

[SQL Functions \[page 323\]](#)

## 1.13.1.7.1 CURRENT DATE Special Value - UltraLite

Returns the current year, month, and day.

### Data Type

DATE

### Remarks

The returned date is based on a reading of the system clock when the SQL statement is executed by the UltraLite runtime. If you use CURRENT DATE with any of the following, all values are based on separate clock readings:

- CURRENT DATE multiple times within the same statement
- CURRENT DATE with CURRENT TIME or CURRENT TIMESTAMP within a single statement
- CURRENT DATE with the NOW function or GETDATE function within a single statement

### Related Information

[Expressions in UltraLite \[page 264\]](#)

[GETDATE Function \[Date and Time\] \[page 401\]](#)

[NOW Function \[Date and Time\] \[page 446\]](#)

## 1.13.1.7.2 CURRENT TIME Special Value - UltraLite

The current hour, minute, second, and fraction of a second.

### Data Type

TIME

## Remarks

The fraction of a second is stored to 6 decimal places. The accuracy of the current time is limited by the accuracy of the system clock.

The returned date is based on a reading of the system clock when the SQL statement is executed by the UltraLite runtime. If you use CURRENT TIME with any of the following, all values are based on separate clock readings:

- CURRENT TIME multiple times within the same statement
- CURRENT TIME with CURRENT DATE or CURRENT TIMESTAMP within a single statement
- CURRENT TIME with the NOW function or GETDATE function within a single statement

## Related Information

[Expressions in UltraLite \[page 264\]](#)

[GETDATE Function \[Date and Time\] \[page 401\]](#)

[NOW Function \[Date and Time\] \[page 446\]](#)

### 1.13.1.7.3 CURRENT TIMESTAMP Special Value - UltraLite

Combines CURRENT DATE and CURRENT TIME to form a TIMESTAMP value containing the year, month, day, hour, minute, second, and fraction of a second.

## Data Type

TIMESTAMP

## Remarks

The fraction of a second is stored to 3 decimal places. The accuracy is limited by the accuracy of the system clock.

Columns declared with DEFAULT CURRENT TIMESTAMP do not necessarily contain unique values.

The information CURRENT TIMESTAMP returns is equivalent to the information returned by the GETDATE and NOW functions.

CURRENT\_TIMESTAMP is equivalent to CURRENT TIMESTAMP.

The returned date is based on a reading of the system clock when the SQL statement is executed by the UltraLite runtime. If you use CURRENT TIMESTAMP with any of the following, all values are based on separate clock readings:

- CURRENT TIMESTAMP multiple times within the same statement
- CURRENT TIMESTAMP with CURRENT DATE or CURRENT TIME within a single statement
- CURRENT TIMESTAMP with the NOW function or GETDATE function within a single statement

## Related Information

[CURRENT TIME Special Value - UltraLite \[page 260\]](#)

[Expressions in UltraLite \[page 264\]](#)

[NOW Function \[Date and Time\] \[page 446\]](#)

[GETDATE Function \[Date and Time\] \[page 401\]](#)

### 1.13.1.7.4 CURRENT UTC TIMESTAMP Special Value - UltraLite

Returns a TIMESTAMP WITH TIME ZONE value that reflects the current UTC time containing the year, month, and day.

## Data Type

DATE

## Remarks

The returned date is based on a reading of the system clock when the SQL statement is executed by the UltraLite runtime.

- CURRENT DATE multiple times within the same statement
- CURRENT DATE with CURRENT TIME or CURRENT TIMESTAMP within a single statement
- CURRENT DATE with the NOW function or GETDATE function within a single statement

## Related Information

[CURRENT TIMESTAMP Special Value - UltraLite \[page 261\]](#)

[Expressions in UltraLite \[page 264\]](#)

[GETDATE Function \[Date and Time\] \[page 401\]](#)

[NOW Function \[Date and Time\] \[page 446\]](#)

### 1.13.1.75 SQLCODE Special Value

Current SQLCODE value at the time the special value was evaluated.

#### Data Type

String

#### Remarks

The SQLCODE value is set after each statement. You can check the SQLCODE to determine if the statement succeeded.

#### Example

Use a SELECT statement to produce an error code for each attempt to fetch a new row from the result set. For example: `SELECT a, b, SQLCODE FROM MyTable.`

#### Related Information

[Expressions in UltraLite \[page 264\]](#)

## 1.13.1.8 Dates and Times in UltraLite

Many of the date and time functions use dates built from date and time parts. UltraLite and SQL Anywhere support the same date parts.

### Related Information

[Specifying Date Parts \[page 328\]](#)

## 1.13.1.9 Expressions in UltraLite

Expressions are formed by combining data, often in the form of column references, with operators or functions.

### ≡ Syntax

```
expression:  
  case-expression  
  | constant  
  | [correlation-name.]column-name  
  | - expression  
  | expression operator expression  
  | ( expression )  
  | function-name ( expression, ... )  
  | if-expression  
  | special value  
  | input-parameter
```

### Parameters

```
case-expression:  
CASE expression  
WHEN expression  
THEN expression, ...  
[ ELSE expression ]  
END
```

```
alternative form of case-expression:  
CASE  
WHEN search-condition  
THEN expression, ...  
[ ELSE expression ]  
END
```

```
constant:  
integer | number | string | host-variable
```



```
special-value:  
  CURRENT { DATE | TIME | TIMESTAMP }  
| NULL  
| SQLCODE  
| SQLSTATE
```

```
if-expression:  
IF condition  
THEN expression  
[ ELSE expression ]  
ENDIF
```

```
input-parameter:  
{ ? | :name [ : indicator-name ] }
```

```
operator:  
{ + | - | * | / | || | % }
```

### In this section:

#### [Constants in Expressions \[page 266\]](#)

In UltraLite, constants are numbers or string literals.

#### [Column Names in Expressions - UltraLite \[page 266\]](#)

An identifier in an expression.

#### [IF Expressions - UltraLite \[page 267\]](#)

Sets a search condition to return a specific subset of data.

#### [CASE Expressions - UltraLite \[page 268\]](#)

Provides conditional SQL expressions.

#### [Aggregate Expressions - UltraLite \[page 269\]](#)

Performs an aggregate computation that the UltraLite runtime does not provide.

#### [Subqueries in Expressions - UltraLite \[page 270\]](#)

A SELECT statement that is nested inside another SELECT statement.

#### [Input Parameters \[page 271\]](#)

Acts as placeholders to allow end-users to supply values to a prepared statement. These user-supplied values are then used to execute the statement.

## Related Information

[SQL Data Types \[page 288\]](#)

[SQL Functions \[page 323\]](#)

### 1.13.1.9.1 Constants in Expressions

In UltraLite, constants are numbers or string literals.

```
☰ Syntax  
' constant '
```

#### Usage

String constants are enclosed in single quotes (').

An apostrophe is represented inside a string by two single quotes in a row (").

#### Example

To use a possessive phrase, type the string literal as follows:

```
'John's database'
```

#### Related Information

[Escape Sequences](#)

### 1.13.1.9.2 Column Names in Expressions - UltraLite

An identifier in an expression.

```
☰ Syntax  
correlation-name.column-name
```

#### Remarks

A column name is preceded by an optional correlation name, which typically is the name of a table.

If a column name is a keyword or has characters other than letters, digits and underscore, it must be surrounded by quotation marks (" "). For example, the following are valid column names:

```
Employees.Name  
address  
"date hired"  
"salary"."date paid"
```

## Related Information

[FROM Clause \[UltraLite\] \[page 550\]](#)

### 1.13.1.9.3 IF Expressions - UltraLite

Sets a search condition to return a specific subset of data.

## Syntax

```
IF search-condition  
THEN expression1  
[ ELSE expression2 ]  
ENDIF
```

## Remarks

For compatibility reasons, this expression can end in either ENDIF or END IF.

This expression returns the following:

- If `search-condition` is TRUE, the IF expression returns `expression1`.
- If `search-condition` is FALSE and an ELSE clause is specified, the IF expression returns `expression2`.
- If `search-condition` is FALSE, and there is no `expression2`, the IF expression returns NULL.
- If `search-condition` is UNKNOWN, the IF expression returns NULL.

## Related Information

[NULL Special Value](#)  
[Search Conditions](#)

## 1.13.1.9.4 CASE Expressions - UltraLite

Provides conditional SQL expressions.

### Syntax 1

```
CASE expression1
WHEN expression2 THEN expression3, ...
[ ELSE expression4 ]
END
```

```
SELECT ID,
( CASE name
  WHEN 'Tee Shirt' THEN 'Shirt'
  WHEN 'Sweatshirt' THEN 'Shirt'
  WHEN 'Baseball Cap' THEN 'Hat'
  ELSE 'Unknown'
END ) as Type
FROM Product
```

### Syntax 2

```
CASE
WHEN search-condition
THEN expression1, ...
[ ELSE expression2 ]
END
```

### Remarks

For compatibility reasons, you can end this expression with either ENDCASE or END CASE.

You can use case expressions anywhere you can use regular expression.

#### Syntax 1

If the expression following the CASE keyword is equal to the expression following the first WHEN keyword, then the expression following the associated THEN keyword is returned. Otherwise the expression following the ELSE keyword is returned, if specified.

For example, the following code uses a case expression as the second clause in a SELECT statement. It selects a row from the Product table where the name column has a value of Sweatshirt.

#### Syntax 2

If the search-condition following the first WHEN keyword is TRUE, the expression following the associate THEN keyword is returned. Otherwise the expression following the ELSE clause is returned, if specified.

## NULLIF function for abbreviated CASE expressions

The NULLIF function provides a way to write some CASE statements in short form. The syntax for NULLIF is as follows:

```
NULLIF ( expression-1, expression-2 )
```

NULLIF compares the values of the two expressions. If the first expression equals the second expression, NULLIF returns NULL. If the first expression does not equal the second expression, NULLIF returns the first expression.

## Example

The following statement uses a CASE expression as the third clause of a SELECT statement to associate a string with a search condition. If the name column's value is **Tee Shirt**, this query returns **Sale**. And if the name column's value is not **Tee Shirt** and the quantity is greater than fifty, it returns **Big Sale**. However, for all others, the query then returns **Regular price**.

```
SELECT ID, name,  
       ( CASE  
         WHEN name='Tee Shirt' THEN 'Sale'  
         WHEN quantity >= 50 THEN 'Big Sale'  
         ELSE 'Regular price'  
       END ) as Type  
FROM Product
```

## 1.13.1.9.5 Aggregate Expressions - UltraLite

Performs an aggregate computation that the UltraLite runtime does not provide.

☰, Syntax

```
SUM( expression )
```

## Remarks

An aggregate expression calculates a single value from a range of rows.

An aggregate expression is one in which either an aggregate function is used, or in which one or more of the operands is an aggregate expression.

When a SELECT statement does not have a GROUP BY clause, the expressions in the SELECT list must either contain all aggregate expressions or no aggregate expressions. When a SELECT statement does have a GROUP BY clause, any non-aggregate expression in the SELECT list must appear in the GROUP BY list.

## Example

For example, the following query computes the total payroll for employees in the employee table. In this query, `SUM( salary )` is an aggregate expression:

```
SELECT SUM( salary )
FROM employee
```

## 1.13.1.9.6 Subqueries in Expressions - UltraLite

A SELECT statement that is nested inside another SELECT statement.

### ⚡ Syntax

A subquery is structured like a regular query.

## Remarks

In UltraLite, you can only use subquery references in the following situations:

- As a table expression in the FROM clause. This form of table expression (also called **derived tables**) must have a derived table name and column names in which values in the SELECT list are fetched.
- To supply values for the EXISTS, ANY, ALL, and IN search conditions.

You can write subqueries about names that are specified before (to the left of) the subquery, sometimes known as outer references to the left. However, you cannot have references to items within subqueries (sometimes known as inner references).

## Example

The following subquery is used to list all product IDs for items that are low in stock (that is, less than 20 items).

```
FROM SalesOrderItems
( SELECT ID
  FROM Products
  WHERE Quantity < 20 )
```

## Related Information

[Use of Subqueries](#)

[SELECT Statement \[UltraLite\] \[page 558\]](#)

## 1.13.1.9.7 Input Parameters

Acts as placeholders to allow end-users to supply values to a prepared statement. These user-supplied values are then used to execute the statement.

### ≡ Syntax

```
{ ? | :name [ : indicator-name ] }
```

### Remarks

Use the placeholder character of ? or the named form in expressions. You can use input parameters whenever you can use a column name or constant.

The precise mechanism used to supply the values to the statement are dependent upon the API you use to create your UltraLite client.

#### Using the named form

The named form of an input parameter has special meaning. In general, `name` is always used to specify multiple locations where an actual value is supplied.

For Embedded SQL applications only, the `indicator-name` supplies the variable into which the null indicator is placed. If you use the named form with the other components, `indicator-name` is ignored.

#### Deducing data types

The data type of the input parameter is deduced when the statement is prepared from one of the following patterns:

- `CAST ( ? AS type )`  
In this case, `type` is a database type specification such as `CHAR(32)`.
- Exactly one operand of a binary operator is an input parameter. The type is deduced to be the type of the operand.

If the type cannot be deduced, UltraLite generates an error. For example:

- `-?`: the operand is unary.
- `? + ?`: both are input parameters.

### Example

The following Embedded SQL statement has two input parameters:

```
INSERT INTO MyTable VALUES ( :v1, :v2, :v1)
```

The first instance of v1 supplies its value to both the v2 and v1 locations in the statement.

## Related Information

[Host Variables \[page 683\]](#)

[Prepared Statements](#)

[Data Modification in UltraLite C++ Using INSERT, UPDATE, and DELETE \[page 650\]](#)

[Data Modification in UltraLite.NET Using INSERT, UPDATE, and DELETE \[page 605\]](#)

### 1.13.1.10 Search Conditions in UltraLite

A search condition is the criteria for a WHERE clause, a HAVING clause, an ON phrase in a join, or an IF expression. A search condition is also called a **predicate**.

#### ☞ Syntax

```
search-condition:  
expression comparison-operator expression  
| expression IS [ NOT ] NULL  
| expression [ NOT ] BETWEEN expression AND expression  
| expression [ NOT ] IN ( expression, ... )  
| expression [ NOT ] IN ( subquery )  
| expression [ NOT ] { ANY | ALL } ( subquery )  
| expression [ NOT ] EXISTS ( subquery )  
| expression [ NOT ] LIKE ( pattern )  
| NOT search-condition  
| search-condition AND search-condition  
| search-condition OR search-condition  
| ( search-condition IS [ NOT ] { TRUE | FALSE | UNKNOWN }
```

```
comparison-operator :  
=  
| >  
| <  
| >=  
| <=  
| <>  
| !=  
| !<  
| !>
```

## Parameters

The different types of search conditions supported by UltraLite include:

- ALL condition



- ANY condition
- BETWEEN condition
- EXISTS condition
- IN condition
- LIKE condition

## Remarks

In UltraLite, search conditions can appear in the:

- WHERE clause
- HAVING clause
- ON phrase
- SQL queries

Search conditions can be used to choose a subset of the rows from a table in a FROM clause in a SELECT statement, or in expressions such as an IF or CASE to select specific values. In UltraLite, every condition evaluates as one of three states: TRUE, FALSE, or UNKNOWN. When combined, these states are referred to as **three-valued logic**. The result of a comparison is UNKNOWN if either value being compared is the NULL value. Search conditions are satisfied only if the result of the condition is TRUE.

### In this section:

#### [Comparison Operators - UltraLite \[page 274\]](#)

Any operator that allows two or more expressions to be compared with in a search condition.

#### [Logical Operators - UltraLite \[page 275\]](#)

Logical operators can be used as search conditions (for example, AND, OR, and NOT), or test the truth or NULL value of an IS expressions.

#### [ALL Search Condition - UltraLite \[page 277\]](#)

Use the ALL condition with comparison operators to compare a single value to the data values produced by the subquery.

#### [ANY Search Condition - UltraLite \[page 278\]](#)

Use the ANY condition with comparison operators to compare a single value to the column of data values produced by the subquery.

#### [BETWEEN Search Condition - UltraLite \[page 279\]](#)

Specifies an inclusive range, in which the lower value and the upper value and the values they delimit are searched for.

#### [EXISTS Search Condition - UltraLite \[page 280\]](#)

Checks whether a subquery produces any rows of query results

#### [IN Search Condition - UltraLite \[page 280\]](#)

Checks membership by searching a value from the main query with another value in the subquery.

#### [LIKE Search Condition - UltraLite \[page 281\]](#)

Checks whether a pattern is found in an expression.

## Related Information

[SQL Data Types \[page 288\]](#)

[SQL Functions \[page 323\]](#)

[Queries](#)

[Pattern Matching Character Strings in the WHERE Clause](#)

[Three-valued Logic](#)

[Subqueries in Expressions - UltraLite \[page 270\]](#)

### 1.13.1.10.1 Comparison Operators - UltraLite

Any operator that allows two or more expressions to be compared with in a search condition.

☞ Syntax

```
expression operator expression
```

## Parameters

Operator	Interpretation
=	equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
!=	not equal to
<>	not equal to
!>	not greater than
!<	not less than

## Remarks

### Comparing dates

In comparing dates, < means earlier and > means later.

### Comparing LONG VARCHAR or LONG BINARY values

UltraLite does not support comparisons using LONG VARCHAR or LONG BINARY values.

### Case-sensitivity

In UltraLite, comparisons are carried out with the same attention to case as the database on which they are operating. By default, UltraLite databases are created as case insensitive.

### NOT operator

The NOT operator negates an expression.

## Example

Either of the following two queries will find all Tee shirts and baseball caps that cost \$10 or less. However, note the difference in position between the negative logical operator (NOT) and the negative comparison operator (!>).

```
SELECT ID, Name, Quantity
FROM Products
WHERE (name = 'Tee Shirt' OR name = 'BaseBall Cap')
AND NOT UnitPrice > 10
```

```
SELECT ID, Name, Quantity
FROM Products
WHERE (name = 'Tee Shirt' OR name = 'BaseBall Cap')
AND UnitPrice !> 10
```

## Related Information

[Logical Operators - UltraLite \[page 275\]](#)

### 1.13.1.10.2 Logical Operators - UltraLite

Logical operators can be used as search conditions (for example, AND, OR, and NOT), or test the truth or NULL value of an IS expressions.

#### Syntax 1

```
condition1 logical-operator condition2
```

## Syntax 2

```
NOT condition
```

## Syntax 3

```
expression IS [ NOT ] { truth-value | NULL }
```

## Remarks

Search conditions can be used to choose a subset of the rows from a table in a FROM clause in a SELECT statement, or in expressions such as an IF or CASE to select specific values. In UltraLite, every condition evaluates as one of three states: TRUE, FALSE, or UNKNOWN. When combined, these states are referred to as **three-valued logic**. The result of a comparison is UNKNOWN if either value being compared is the NULL value. Search conditions are satisfied only if the result of the condition is TRUE.

### AND

The combined condition is TRUE if both conditions are TRUE, FALSE if either condition is FALSE, and UNKNOWN otherwise.

```
condition1 AND condition2
```

### OR

The combined condition is TRUE if either condition is TRUE, FALSE if both conditions are FALSE, and UNKNOWN otherwise.

### NOT

The NOT condition is TRUE if `condition` is FALSE, FALSE if `condition` is TRUE, and UNKNOWN if `condition` is UNKNOWN.

### IS

The condition is TRUE if the `expression` evaluates to the supplied `truth-value`, which must be one of TRUE, FALSE, or UNKNOWN. Otherwise, the value is FALSE.

## Example

The IS NULL condition is satisfied if the column contains a NULL value. If you use the IS NOT NULL operator, the condition is satisfied when the column contains a value that is not NULL. This example shows an IS NULL condition: WHERE paid\_date IS NULL.

## Related Information

[Three-valued Logic](#)

[Comparison Operators - UltraLite \[page 274\]](#)

### 1.13.1.10.3 ALL Search Condition - UltraLite

Use the ALL condition with comparison operators to compare a single value to the data values produced by the subquery.

☞ Syntax

```
expression compare [ NOT ] ALL ( subquery )
```

## Parameters

```
compare:  
= | > | < | >= | <= | <> | != | !< | !>
```

## Remarks

UltraLite uses the specified comparison operator to compare the test value to each data value in the result set. If all the comparisons yield TRUE results, the ALL test returns TRUE.

## Example

Find the order and customer IDs of those orders placed after all products of order #2001 were shipped.

```
SELECT ID, CustomerID  
FROM SalesOrders  
WHERE OrderDate > ALL (  
  SELECT ShipDate  
  FROM SalesOrderItems  
  WHERE ID=2001)
```

## Related Information

[Subqueries and the ALL Test](#)

## 1.13.1.10.4 ANY Search Condition - UltraLite

Use the ANY condition with comparison operators to compare a single value to the column of data values produced by the subquery.

### Syntax 1

```
expression compare [ NOT ] ANY ( subquery )
```

### Syntax 2

```
expression = ANY ( subquery )
```

### Parameters

```
compare:  
= | > | < | >= | <= | <> | != | !< | !>
```

### Remarks

UltraLite uses the specified comparison operator to compare the test value to each data value in the column. If any of the comparisons yields a TRUE result, the ANY test returns TRUE.

#### Syntax 1

is TRUE if *expression* is equal to any of the values in the result of the subquery, and FALSE if the expression is not NULL and does not equal any of the values returned by the subquery. The ANY condition is UNKNOWN if *expression* is the NULL value, unless the result of the subquery has no rows, in which case the condition is always FALSE.

## Example

Find the order and customer IDs of those orders placed after the first product of the order #2005 was shipped.

```
SELECT ID, CustomerID
FROM SalesOrders
WHERE OrderDate > ANY (
  SELECT ShipDate
  FROM SalesOrderItems
  WHERE ID=2005)
```

## Related Information

[Subqueries and the ANY Test](#)

[Comparison Operators - UltraLite \[page 274\]](#)

### 1.13.1.10.5 BETWEEN Search Condition - UltraLite

Specifies an inclusive range, in which the lower value and the upper value and the values they delimit are searched for.

#### ☞ Syntax

```
expression [ NOT ] BETWEEN start-expression AND end-expression
```

## Remarks

The BETWEEN condition can evaluate to TRUE, FALSE, or UNKNOWN. Without the NOT keyword, the condition evaluates as TRUE if *expression* is between *start-expression* and *end-expression*. The NOT keyword reverses the meaning of the condition, but leaves UNKNOWN unchanged.

The BETWEEN condition is equivalent to a combination of two inequalities:

```
[ NOT ] ( expression >= start-expression
          AND expression <= end-expression )
```

## Example

List all the products less expensive than \$10 or more expensive than \$15.

```
SELECT Name, UnitPrice
```

```
FROM Products
WHERE UnitPrice NOT BETWEEN 10 AND 15
```

### 1.13.1.10.6 EXISTS Search Condition - UltraLite

Checks whether a subquery produces any rows of query results

#### ☞ Syntax

```
[ NOT ] EXISTS ( subquery )
```

#### Remarks

The EXISTS condition is TRUE if the subquery result contains at least one row, and FALSE if the subquery result does not contain any rows. The EXISTS condition cannot be UNKNOWN.

You can reverse the logic of the EXISTS condition by using the NOT EXISTS form. In this case, the test returns TRUE if the subquery produces no rows, and FALSE otherwise.

#### Example

List the customers who placed orders after July 13, 2001.

```
SELECT GivenName, Surname
FROM Customers
WHERE EXISTS (
  SELECT *
  FROM SalesOrders
  WHERE (OrderDate > '2001-07-13') AND
        (Customers.ID = SalesOrders.CustomerID))
```

### 1.13.1.10.7 IN Search Condition - UltraLite

Checks membership by searching a value from the main query with another value in the subquery.

#### ☞ Syntax

```
expression [ NOT ] IN
{ ( subquery ) | ( value-expr, ... ) }
```



## Parameters

`value-expr` are expressions that take on a single value, which may be a string, a number, a date, or any other SQL data type.

## Remarks

An IN condition, without the NOT keyword, evaluates according to the following rules:

- TRUE if `expression` is not NULL and equals at least one of the values.
- UNKNOWN if `expression` is NULL and the values list is not empty, or if at least one of the values is NULL and `expression` does not equal any of the other values.
- FALSE if `expression` is NULL and `subquery` returns no values; or if `expression` is not NULL, none of the values are NULL, and `expression` does not equal any of the values.

You can reverse the logic of the IN condition by using the NOT IN form.

The following search condition `expression IN ( values )` is identical to the search condition `expression = ANY ( values )`. The search condition `expression NOT IN ( values )` is identical to the search condition `expression <> ALL ( values )`.

## Example

Select the company name and state for customers who live in the following Canadian provinces: Ontario, Manitoba, and Quebec.

```
SELECT CompanyName , Province
FROM Customers
WHERE State IN( 'ON', 'MB', 'PQ')
```

### 1.13.1.10.8 LIKE Search Condition - UltraLite

Checks whether a pattern is found in an expression.

#### ☞ Syntax

The syntax for the LIKE search condition is as follows:

```
expression [ NOT ] LIKE pattern
```

## Parameters

### **expression**

The string to be searched.

### **pattern**

The pattern to search for within *expression*.

## Remarks

The LIKE search condition attempts to match *expression* with *pattern* and evaluates to TRUE, FALSE, or UNKNOWN. The search condition evaluates to TRUE if *expression* matches *pattern* (assuming NOT was not specified). If either *expression* or *pattern* is the NULL value, the search condition evaluates to UNKNOWN.

The NOT keyword reverses the meaning of the search condition, but leaves UNKNOWN unchanged.

*expression* and *pattern* are interpreted as CHAR strings. *pattern* can contain any number of the supported wildcards from the following table:

Wildcard	Matches
_ (underscore)	Any one character. For example, a_ matches ab and ac, but not a.
% (percent)	Any string of zero or more characters. For example, bl% matches bl and bla.
[]	Any single character in the specified range or set. For example, T[o i]m matches Tom or Tim.
[^]	Any single character <i>not</i> in the specified range or set. For example, M[^c] matches Mb and Md, but not Mc.

## Different Ways to Use the LIKE Search Condition

To search for	Example	Additional information
One of a set of characters	LIKE 'sm[iy]th'	A set of characters to look for is specified by listing the characters inside square brackets. In this example, the search condition matches <i>smith</i> and <i>smyth</i> .

To search for	Example	Additional information
One of a range of characters	LIKE ' [a-r] ough '	<p>A range of characters to look for is specified by giving the ends of the range inside square brackets, separated by a hyphen. In this example, the search condition matches bough and rough, but not tough.</p> <p>The range of characters [a-z] is interpreted as "greater than or equal to a, and less than or equal to z", where the greater than and less than operations are carried out within the collation of the database.</p> <p>The lower end of the range must precede the higher end of the range. For example, [z-a] does not match anything because no character matches the [z-a] range.</p>
Ranges and sets combined	... LIKE ' [a-rt] ough '	<p>You can combine ranges and sets within square brackets. In this example, ... LIKE ' [a-rt] ough ' matches bough, rough, and tough.</p> <p>The pattern [a-rt] is interpreted as exactly one character that is either in the range a to r inclusive, or is t.</p>
One character not in a range	... LIKE ' [^a-r] ough '	<p>The caret character (^) is used to specify a range of characters that is excluded from a search. In this example, LIKE ' [^a-r] ough ' matches the string tough, but not the strings rough or bough.</p> <p>The caret negates the rest of the contents of the brackets. For example, the bracket [^a-rt] is interpreted as exactly one character that is not in the range a to r inclusive, and is not t.</p>

To search for	Example	Additional information
Search patterns with trailing blanks	'90 ', '90[ ]' and '90_'	When your search pattern includes trailing blanks, the database server matches the pattern only to values that contain blanks. It does not blank pad strings. For example, the patterns '90 ', '90[ ]', and '90_' match the expression '90 ', but do not match the expression '90', even if the value being tested is in a CHAR or VARCHAR column that is three or more characters in width.

## Special Cases of Ranges and Sets

Any single character in square brackets means that character. For example, [a] matches just the character a. [^] matches just the caret character, [%] matches just the percent character (the percent character does not act as a wildcard in this context), and [\_] matches just the underscore character. Also, [ ] matches just the character [.

- The pattern [a-] matches either of the characters a or -.
- The pattern [ ] is never matched and always returns no rows.
- The patterns [ or [abp-q return syntax errors because they are missing the closing bracket.
- You cannot use wildcards inside square brackets. The pattern [a%b] finds one of a, %, or b.
- You cannot use the caret character to negate ranges except as the first character in the bracket. The pattern [a^b] finds one of a, ^, or b.

## Case Sensitivity and How Comparisons are Performed

If the database collation is case sensitive, the search condition is also case sensitive. To perform a case insensitive search with a case sensitive collation, you must include upper and lower characters. For example, the following search condition evaluates to true for the strings Bough, rough, and TOUGH:

```
LIKE '[a-zA-Z][oO][uU][gG][hH]'
```

## Example

The following search condition returns TRUE for any row where column-name starts with the letter a and has the letter b as its second last character:

```
SELECT * FROM table-name WHERE column-name LIKE 'a%b_'
```

## 1.13.1.11 Operators in UltraLite

Operators are used to compute values, which may in turn be used as operands in a higher-level expression.

UltraLite SQL supports the following types of operators:

- Comparison operators evaluate and return a result using one (unary) or two (binary) comparison operands. Comparisons result in the usual three logical values: true, false, and unknown.
- Arithmetic operators evaluate and return a result set for all floating-point, decimal, and integer numbers.
- String operators concatenate two string values together. For example, "my" + "string" returns the string "my string".
- Bitwise operators evaluate and turn specific bits on or off within the internal representation of an integer.
- Logical operators evaluate search conditions. Logical evaluations result in the usual three logical values: true, false, and unknown.

The normal precedence of operations applies.

### In this section:

#### [Arithmetic Operators - UltraLite \[page 285\]](#)

Arithmetic operators allow you to perform calculations.

#### [String Operators - UltraLite \[page 286\]](#)

String operators allow you to concatenate strings, except for LONGVARCHAR and LONGBINARY data types.

#### [Bitwise Operators - UltraLite \[page 287\]](#)

Bitwise operators perform bit manipulations between two expressions. The following operators can be used on integer data types in UltraLite.

#### [Operator Precedence - UltraLite \[page 287\]](#)

The order of operators in an expression is significant because it impacts the order in which the sub-expressions are evaluated.

## Related Information

[Comparison Operators - UltraLite \[page 274\]](#)

[Logical Operators - UltraLite \[page 275\]](#)

## 1.13.1.11.1 Arithmetic Operators - UltraLite

Arithmetic operators allow you to perform calculations.

### **expression + expression**

Addition. If either expression is NULL, the result is NULL.

### **expression - expression**

Subtraction. If either expression is NULL, the result is NULL.

**- expression**

Negation. If the expression is NULL, the result is NULL.

**expression \* expression**

Multiplication. If either expression is NULL, the result is NULL.

**expression / expression**

Division. If either expression is NULL or if the second expression is 0, the result is NULL.

**expression % expression**

Modulo finds the integer remainder after a division involving two whole numbers. For example,  $21 \% 11 = 10$  because 21 divided by 11 equals 1 with a remainder of 10. If either expression is NULL, the result is NULL.

## Related Information

[Computed Values in the SELECT List](#)

### 1.13.1.11.2 String Operators - UltraLite

String operators allow you to concatenate strings, except for LONGVARCHAR and LONGBINARY data types.

**expression || expression**

String concatenation (two vertical bars). If either string is NULL, it is treated as the empty string for concatenation.

**expression + expression**

Alternative string concatenation. When using the + concatenation operator, you must ensure the operands are explicitly set to character data types rather than relying on implicit data conversion.

For example, the following query returns the integer value [579](#):

```
SELECT 123 + 456
```

However, the following query returns the character string [123456](#):

```
SELECT '123' + '456'
```

You can use the CAST or CONVERT functions to explicitly convert data types.

### 1.13.1.11.3 Bitwise Operators - UltraLite

Bitwise operators perform bit manipulations between two expressions. The following operators can be used on integer data types in UltraLite.

Operator	Description
&	bitwise AND
	bitwise OR
^	bitwise exclusive OR
~	bitwise NOT

The bitwise operators &, |, and ~ are not interchangeable with the logical operators AND, OR, and NOT. The bitwise operators operate on integer values using the bit representation of the values.

#### Example

The following statement selects rows in which the specified bits are set.

```
SELECT *
FROM tableA
WHERE (options & 0x0101) <> 0
```

### 1.13.1.11.4 Operator Precedence - UltraLite

The order of operators in an expression is significant because it impacts the order in which the sub-expressions are evaluated.

Expressions in parentheses are evaluated first, then multiplication and division before addition and subtraction. String concatenation happens after addition and subtraction. The operators at the top of the list are evaluated before those at the bottom of the list.

#### i Note

Make the order of operation explicit. For example, use parentheses, rather than relying on an operator precedence, when specifying more than one operator in an expression.

1. names, functions, constants, IF expressions, CASE expressions
2. ()
3. unary operators (operators that require a single operand): +, -
4. ~
5. &, |, ^
6. \*, /, %

7. +, -
8. ||
9. comparisons: >, <, <>, !=, <=, >=, [ NOT ] BETWEEN, [ NOT ] IN, [ NOT ] LIKE
10. comparisons: IS [NOT] TRUE, FALSE, UNKNOWN
11. NOT
12. AND
13. OR

### 1.13.1.12 Variables in UltraLite

You cannot use SQL variables (including global variables) in UltraLite applications.

## 1.13.2 SQL Data Types

UltraLite supports a subset of the data types available in SQL Anywhere.

### In this section:

#### [Character Data Types \[page 288\]](#)

Character data types store strings of letters, numbers, and other symbols.

#### [Numeric Data Types \[page 294\]](#)

Numeric data types store numerical data.

#### [Date and Time Data Types \[page 306\]](#)

Date values can be output in full century format, and the internal storage of dates always explicitly includes the century portion of a year value.

#### [Binary Data Types \[page 315\]](#)

Binary data types store binary data, including images and other types of information that are not interpreted by the database.

### 1.13.2.1 Character Data Types

Character data types store strings of letters, numbers, and other symbols.

There are classes of character data types and some domains defined using those types:

#### **CHAR, VARCHAR, LONG VARCHAR**

Character data stored in a single- or multibyte character set, often chosen to correspond most closely to the primary language or languages stored in the database.

#### **NCHAR, NVARCHAR, LONG NVARCHAR**

Character data stored in the UTF-8 Unicode encoding. All Unicode code points can be stored using these types, regardless of the primary language or languages stored in the database.



## TEXT, UNIQUEIDENTIFIER, XML

Domains based on other character data types.

**UltraLite:** UltraLite supports the CHAR, VARCHAR, and LONG VARCHAR data types, which are stored in a single- or multi- byte character set, and are often chosen to correspond most closely to the primary language or languages stored in the database.

## Storage

All character data values are stored in the same manner. By default, values up to 128 bytes are stored in a single piece. Values longer than 128 bytes are stored with a 4-byte prefix kept locally on the database page and the full value stored in one or more other database pages. These default sizes are controlled by the INLINE and PREFIX clauses of the CREATE TABLE statement.

**UltraLite:** Fixed character types, such as VARCHAR, are embedded in the row whereas long character types, such as LONG VARCHAR, are stored separately. Consider your page size when creating a table with many columns of large fixed types. A full row must fit on a page, and fixed character column types are stored with a row. For example, a database created with a page size of 1000 cannot hold character values larger than 1000 because they cannot fit on the page.

### In this section:

[CHAR Data Type \[page 289\]](#)

The CHAR data type stores character data, up to 32767 bytes.

[LONG VARCHAR Data Type \[page 291\]](#)

The LONG VARCHAR data type stores character data of arbitrary length.

[VARCHAR Data Type \[page 292\]](#)

The VARCHAR data type stores character data, up to 32767 bytes.

## Related Information

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

### 1.13.2.1.1 CHAR Data Type

The CHAR data type stores character data, up to 32767 bytes.

#### ☰ Syntax

```
CHAR [ ( max-length [ CHAR | CHARACTER ] ) ]
```

#### UltraLite:

```
CHAR [ ( max-length ) ]
```

## Parameters

### max-length

The maximum length of the string. If byte-length semantics are used (CHAR or CHARACTER is not specified as part of the length), then the length is in bytes, and the length must be in the range 1 to 32767. If the length is not specified, then it is 1.

If character-length semantics are used (CHAR or CHARACTER is specified as part of the length), then the length is in characters, and you must specify `max-length`. `max-length` can be a maximum of 32767 characters.

## Remarks

Multibyte characters can be stored as CHAR, but the declared length refers to bytes, not characters, unless character-length semantics are used.

CHAR can also be specified as CHARACTER. Regardless of which syntax is used, the data type is described as CHAR.

CHAR is semantically equivalent to VARCHAR, although they are different types. CHAR is a variable-length type. In other relational database management systems, CHAR is a fixed-length type, and data is padded with blanks to `max-length` bytes of storage. SQL Anywhere does not blank-pad stored character data.

How CHAR columns are described depends on the client interface, the character sets used, and if character-length semantics are used. For example, in Embedded SQL the described length is the maximum number of bytes in the client character set. If the described length would be more than 32767 bytes, the column is described as type DT\_LONGVARCHAR. The following table shows some Embedded SQL examples and the results returned when a DESCRIBE is performed:

Type being described	Database character set	Client character set	Result of DESCRIBE
CHAR(10)	Windows-1252	Windows-1252	DT_FIXCHAR length 10
CHAR(10)	UTF-8	UTF-8	DT_FIXCHAR length 10
CHAR(10)	Windows-1252	UTF-8	DT_FIXCHAR length 30
CHAR(20000)	Windows-31J	UTF-8	DT_LONGVARCHAR
CHAR(10 CHAR )	Windows-1252	Windows-1252	DT_FIXCHAR length 10
CHAR(10 CHAR )	UTF-8	UTF-8	DT_FIXCHAR length 40

For ODBC, CHAR is described as either SQL\_CHAR or SQL\_VARCHAR depending on the `odbc_distinguish_char_and_varchar` option.

**UltraLite:** CHAR is a domain, implemented as VARCHAR.

## Standards

### ANSI/ISO SQL Standard

Compatible with the ANSI/ISO SQL Standard. In the standard, character-length semantics are the default, whereas in the software, byte-length semantics are the default. There are minor inconsistencies with the SQL standard due to case-insensitive collation support and the software's support for blank-padding.

The ANSI/ISO SQL Standard supports explicit character- or byte-length semantics as SQL Language Feature T061.

## Related Information

[VARCHAR Data Type \[page 292\]](#)

### 1.13.2.1.2 LONG VARCHAR Data Type

The LONG VARCHAR data type stores character data of arbitrary length.

☰ Syntax

*LONG VARCHAR*

## Remarks

The maximum size in bytes is 2 GB minus 1 ( $2^{31} - 1$ ) or 2 147 483 647.

Multibyte characters can be stored as LONG VARCHAR, but the length is in bytes, not characters.

#### UltraLite:

- You can cast strings to/from LONG VARCHAR data.
- LONG VARCHAR data cannot be concatenated.
- LONG VARCHAR columns can be included in the result set of a SELECT query.
- Indexes cannot be created on a LONG VARCHAR type.
- A LONG VARCHAR type can only be used in the LENGTH and CAST functions.
- Conditions in SQL statements, such as in the WHERE clause, cannot operate on LONG VARCHAR columns.
- Only INSERT, UPDATE, and DELETE operations are allowed on LONG VARCHAR column.

## Standards

### ANSI/ISO SQL Standard

Large object support is SQL Language Feature T041.

## Related Information

[CHAR Data Type \[page 289\]](#)

[VARCHAR Data Type \[page 292\]](#)

[LENGTH Function \[String\] \[page 417\]](#)

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

### 1.13.2.1.3 VARCHAR Data Type

The VARCHAR data type stores character data, up to 32767 bytes.

#### ☰ Syntax

```
VARCHAR [ ( max-length [ CHAR | CHARACTER ] ) ]
```

#### UltraLite:

```
VARCHAR [ ( max-length ) ]
```

## Parameters

### max-length

The maximum length of the string. This default value is 1.

If byte-length semantics are used (CHAR or CHARACTER is not specified as part of the length), then the length is in bytes, and the length must be in the range of 1 to 32767.

If character-length semantics are used (CHAR or CHARACTER is specified as part of the length), then the length is in characters, and you must specify `max-length`. `max-length` can be a maximum of 32767 characters.

**UltraLite:** UltraLite databases only support byte-length semantics. A non-English character can require up to 3 bytes of storage.

## Remarks

Multibyte characters can be stored as VARCHAR, but the declared length refers to bytes, not characters, unless character-length semantics are used.

**UltraLite:** UltraLite compacts data as much as possible. When a VARCHAR value does not require the number of bytes specified by `max-length`, then only the number of bytes needed to store the value is used. When evaluating expressions, the maximum length for a temporary character value is 2048 bytes.

## ⚠ Caution

### UltraLite:

Although it is possible to create a table with a VARCHAR column where the `max-length` exceeds the page size, an error occurs if you insert a value with a length exceeding that page size.

VARCHAR can also be specified as CHAR VARYING or CHARACTER VARYING. Regardless of which syntax is used, the data type is described as VARCHAR.

VARCHAR is semantically equivalent to CHAR, although they are different types. In SQL Anywhere, CHAR is a variable-length type. In other relational database management systems, CHAR is a fixed-length type, and data is padded with blanks to `max-length` bytes of storage. SQL Anywhere does not blank-pad stored character data.

How VARCHAR columns are described depends on the client interface, the character sets used, and if character-length semantics are used. For example, in Embedded SQL the described length is the maximum number of bytes in the client character set. If the described length would be more than 32767 bytes, the column is described as type DT\_LONGVARCHAR. The following table shows some Embedded SQL examples and the results returned when a DESCRIBE is performed:

Type being described	Database character set	Client character set	Result of DESCRIBE
VARCHAR(10)	Windows-1252	Windows-1252	DT_VARCHAR length 10
VARCHAR(10)	UTF-8	UTF-8	DT_VARCHAR length 10
VARCHAR(10)	Windows-1252	UTF-8	DT_VARCHAR length 30
VARCHAR(20000)	Windows-31J	UTF-8	DT_LONGVARCHAR
VARCHAR(10 CHAR)	Windows-1252	Windows-1252	DT_VARCHAR length 10
VARCHAR(10 CHAR)	UTF-8	UTF-8	DT_VARCHAR length 40

For ODBC, VARCHAR is described as SQL\_VARCHAR.

## Standards

### ANSI/ISO SQL Standard

Compatible with the ANSI/ISO SQL Standard. In the standard, character-length semantics are the default, whereas in the software, byte-length semantics are the default. There are minor inconsistencies with the SQL standard due to case-insensitive collation support and support for blank-padding by the software.

The ANSI/ISO SQL Standard supports explicit character- or byte-length semantics as SQL Language Feature T061.

## Related Information

[CHAR Data Type \[page 289\]](#)

[LONG VARCHAR Data Type \[page 291\]](#)

## 1.13.2.2 Numeric Data Types

Numeric data types store numerical data.

The NUMERIC and DECIMAL data types, and the various INTEGER data types, are sometimes called **exact** numeric data types, in contrast to the **approximate** numeric data types FLOAT, DOUBLE, and REAL.

The exact numeric data types are those for which precision and scale values can be specified, while approximate numeric data types are stored in a predefined manner. *Only exact numeric data is guaranteed accurate to the least significant digit specified after an arithmetic operation.*

Data type lengths and precision of less than one are not allowed.

### Compatibility

Be careful when using default precision and scale settings for NUMERIC and DECIMAL data types because these settings could be different in other database solutions. The default precision is 30 and the default scale is 6.

The FLOAT ( *p* ) data type is a synonym for REAL or DOUBLE, depending on the value of *p*. For SQL Anywhere, the cutoff is platform-dependent, but on all platforms the cutoff value is greater than 15.

Only the NUMERIC data type with scale = 0 can be used for the Transact-SQL identity column. Avoid default precision and scale settings for NUMERIC and DECIMAL data types, because these are different between SQL Anywhere and Adaptive Server Enterprise. In SQL Anywhere, the default precision is 30 and the default scale is 6. In Adaptive Server Enterprise, the default precision is 18 and the default scale is 0.

#### In this section:

##### [BIGINT Data Type \[page 295\]](#)

The BIGINT data type stores BIGINTs, which are integers requiring 8 bytes of storage.

##### [BIT Data Type \[page 296\]](#)

The BIT data type stores a bit (0 or 1).

##### [DECIMAL Data Type \[page 297\]](#)

The DECIMAL data type is a decimal number with *precision* total digits and with *scale* digits after the decimal point.

##### [DOUBLE Data Type \[page 298\]](#)

The DOUBLE data type stores double-precision floating-point numbers.

##### [FLOAT Data Type \[page 299\]](#)

The FLOAT data type stores a floating-point number, which can be single or double precision.

##### [INTEGER Data Type \[page 300\]](#)

The INTEGER data type stores integers that require 4 bytes of storage.

##### [NUMERIC Data Type \[page 301\]](#)

The NUMERIC data type stores decimal numbers with *precision* total digits and with *scale* digits after the decimal point.

##### [REAL Data Type \[page 303\]](#)

The REAL data type stores single-precision floating-point numbers stored in 4 bytes.

[SMALLINT Data Type \[page 304\]](#)

The SMALLINT data type stores integers that require 2 bytes of storage.

[TINYINT Data Type \[page 305\]](#)

The TINYINT data type stores unsigned integers requiring 1 byte of storage.

### 1.13.2.2.1 BIGINT Data Type

The BIGINT data type stores BIGINTs, which are integers requiring 8 bytes of storage.

☰ Syntax

```
[ UNSIGNED ] BIGINT
```

#### Remarks

The BIGINT data type is an exact numeric data type: its accuracy is preserved after arithmetic operations.

A BIGINT value requires 8 bytes of storage.

The range for BIGINT values is  $-2^{63}$  to  $2^{63} - 1$ , or -9223372036854775808 to 9223372036854775807.

The range for UNSIGNED BIGINT values is 0 to  $2^{64} - 1$ , or 0 to 18446744073709551615.

By default, the data type is signed.

When converting a string to a BIGINT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

#### Standards

##### ANSI/ISO SQL Standard

SQL Language Feature T071.

##### MySQL

The UNSIGNED keyword may follow BIGINT.

#### Related Information

[Numeric Functions \[page 332\]](#)

[Aggregate Functions \[page 324\]](#)

[BIT Data Type \[page 296\]](#)

[VARCHAR Data Type \[page 292\]](#)

[SMALLINT Data Type \[page 304\]](#)

[TINYINT Data Type \[page 305\]](#)

## 1.13.2.2.2 BIT Data Type

The BIT data type stores a bit (0 or 1).

☞ Syntax

*BIT*

### Remarks

BIT is an integer type that can store the values 0 or 1.

By default, the BIT data type does not allow NULL.

When converting a string to a BIT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is not 0 or 1.

A BIT value requires 1 byte of storage.

**UltraLite:** A BIT value requires 1 bit of storage.

### Standards

#### ANSI/ISO SQL Standard

Not in the standard.

### Related Information

[Numeric Functions \[page 332\]](#)

[Aggregate Functions \[page 324\]](#)

[BIGINT Data Type \[page 295\]](#)

[VARCHAR Data Type \[page 292\]](#)



[SMALLINT Data Type \[page 304\]](#)

[TINYINT Data Type \[page 305\]](#)

## 1.13.2.2.3 DECIMAL Data Type

The DECIMAL data type is a decimal number with *precision* total digits and with *scale* digits after the decimal point.

☰ Syntax

```
DECIMAL [ ( precision [ , scale ] ) ]
```

### Parameters

#### **precision**

An integer expression between 1 and 127, inclusive, that specifies the number of digits in the expression. The default setting is 30.

#### **scale**

An integer expression between 0 and 127, inclusive, that specifies the number of digits after the decimal point. The scale value should always be less than, or equal to, the precision value.

If precision and scale are both omitted, the default scale is 6. If precision is specified but scale is omitted, the default scale is 0.

**UltraLite:** Change the defaults by setting the appropriate creation parameter.

### Remarks

The DECIMAL data type is an exact numeric data type; its accuracy is preserved to the least significant digit after arithmetic operations.

The number of bytes required to store a decimal number can be estimated as

```
2 + INT(((precision - scale) + 1) / 2) + INT((scale + 1) / 2);
```

The INT function takes the integer portion of its argument. The storage is based on the value being stored, not on the maximum precision and scale allowed in the column.

If you are using a precision of 20 or less and a scale of 0, it may be possible to use one of the integer data types (BIGINT, INTEGER, SMALLINT, or TINYINT) instead. Integer values require less storage space than NUMERIC and DECIMAL values with a similar number of significant digits. Operations on integer values, such as fetching or inserting, and arithmetic operators, typically perform better than operations on NUMERIC and DECIMAL values.

## i Note

If you create a column or variable of a DECIMAL data type with a precision or scale that exceeds the precision and scale settings for the database, values are truncated to the database settings. So, if you notice truncated values in a column or variable defined as DECIMAL, check that precision and scale do not exceed the database option settings.

DECIMAL can also be specified as DEC. Regardless of which syntax is used, the data type is described as DECIMAL. DECIMAL is semantically equivalent to NUMERIC.

## Standards

### ANSI/ISO SQL Standard

Core Feature.

## Example

```
DECLARE d1 DECIMAL; // the default scale is 6
DECLARE d2 DECIMAL ( 20 ); // the default scale is 0
```

## Related Information

[Numeric Functions \[page 332\]](#)

[Aggregate Functions \[page 324\]](#)

[FLOAT Data Type \[page 299\]](#)

[REAL Data Type \[page 303\]](#)

[DOUBLE Data Type \[page 298\]](#)

[NUMERIC Data Type \[page 301\]](#)

[UltraLite Precision Creation Option \[page 168\]](#)

[UltraLite scale Creation Option \[page 169\]](#)

## 1.13.2.2.4 DOUBLE Data Type

The DOUBLE data type stores double-precision floating-point numbers.

### ☰ Syntax

*DOUBLE*

## Remarks

The DOUBLE data type is an approximate numeric data type and subject to rounding errors after arithmetic operations. The approximate nature of DOUBLE values means that queries using equalities should generally be avoided when comparing DOUBLE values.

DOUBLE values require 8 bytes of storage.

The range of values is -1.79769313486231e+308 to 1.79769313486231e+308, with numbers close to zero as small as 2.22507385850721e-308. Values held as DOUBLE are accurate to 15 significant digits, but may be subject to rounding errors beyond the fifteenth digit.

## Standards

ANSI/ISO SQL Standard

Core Feature

## Related Information

[Numeric Functions \[page 332\]](#)

[Aggregate Functions \[page 324\]](#)

[FLOAT Data Type \[page 299\]](#)

[REAL Data Type \[page 303\]](#)

[DECIMAL Data Type \[page 297\]](#)

[NUMERIC Data Type \[page 301\]](#)

### 1.13.2.2.5 FLOAT Data Type

The FLOAT data type stores a floating-point number, which can be single or double precision.

☞ Syntax

```
FLOAT [ ( precision ) ]
```

## Parameters

**precision**

An integer expression that specifies the number of bits in the mantissa, the decimal part of a logarithm. For example, in the number 5.63428, the mantissa is 0.63428. The IEEE standard 754 floating-point precision is as follows:

Supplied precision value	Decimal precision	Equivalent SQL data type	Storage size
1-24	7 decimal digits	REAL	4 bytes
25-53	15 decimal digits	DOUBLE	8 bytes

## Remarks

When a column is created using the FLOAT ( *precision* ) data type, columns on all platforms are guaranteed to hold the values to at least the specified minimum precision. REAL and DOUBLE do not guarantee a platform-independent minimum precision.

If *precision* is not supplied, the FLOAT data type is a single-precision floating-point number, equivalent to the REAL data type, and requires 4 bytes of storage.

If *precision* is supplied, the FLOAT data type is either single or double precision, depending on the value of precision specified. The cutoff between REAL and DOUBLE is platform-dependent. Single-precision FLOAT values require 4 bytes of storage, and double-precision FLOAT values require 8 bytes.

The FLOAT data type is an approximate numeric data type. It is subject to rounding errors after arithmetic operations. The approximate nature of FLOAT values means that queries using equalities should be avoided when comparing FLOAT values.

## Standards

### ANSI/ISO SQL Standard

Core Feature.

## 1.13.2.2.6 INTEGER Data Type

The INTEGER data type stores integers that require 4 bytes of storage.

≡ Syntax

```
[ UNSIGNED ] INTEGER
```

## Remarks

The INTEGER data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

If you specify UNSIGNED, the integer can never be assigned a negative number. By default, the data type is signed.

The range for INTEGER values is  $-2^{31}$  to  $2^{31} - 1$ , or -2147483648 to 2147483647.

The range for UNSIGNED INTEGER values is 0 to  $2^{32} - 1$ , or 0 to 4294967295.

When converting a string to an INTEGER, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

## Standards

### ANSI/ISO SQL Standard

Core Feature. However, the UNSIGNED keyword is not in the standard.

### MySQL

The UNSIGNED keyword may follow INTEGER.

## Related Information

[Numeric Functions \[page 332\]](#)

[Aggregate Functions \[page 324\]](#)

[BIGINT Data Type \[page 295\]](#)

[BIT Data Type \[page 296\]](#)

[SMALLINT Data Type \[page 304\]](#)

[TINYINT Data Type \[page 305\]](#)

## 1.13.2.2.7 NUMERIC Data Type

The NUMERIC data type stores decimal numbers with `precision` total digits and with `scale` digits after the decimal point.

☞ Syntax

```
NUMERIC [ ( precision [ , scale ] ) ]
```

## Parameters

### precision

An integer expression between 1 and 127, inclusive, that specifies the number of digits in the expression. The default setting is 30.

### scale

An integer expression between 0 and 127, inclusive, that specifies the number of digits after the decimal point. The scale value should always be less than or equal to the precision value. The default setting is 6.

## Remarks

The NUMERIC data type is an exact numeric data type; its accuracy is preserved to the least significant digit after arithmetic operations.

**UltraLite:** NUMERIC is a domain, implemented as DECIMAL.

The number of bytes required to store a decimal number can be estimated as

```
2 + INT( (BEFORE+1)/2 ) + INT( (AFTER+1)/2 );
```

The INT function takes the integer portion of its argument, and BEFORE and AFTER are the number of significant digits before and after the decimal point. The storage is based on the value being stored, not on the maximum precision and scale allowed in the column.

If you are using a precision of 20 or less and a scale of 0, it may be possible to use one of the integer data types (BIGINT, INTEGER, SMALLINT, or TINYINT) instead. Integer values require less storage space than NUMERIC and DECIMAL values with a similar number of significant digits. Operations on integer values, such as fetching or inserting, and arithmetic operators, typically perform better than operations on NUMERIC and DECIMAL values.

NUMERIC is semantically equivalent to DECIMAL.

### i Note

If you create a column or variable of a NUMERIC data type with a precision or scale that exceeds the precision and scale settings for the database, values are truncated to the database settings. So, if you notice truncated values in a column or variable defined as NUMERIC, check that precision and scale do not exceed the database option settings.

## Standards

### ANSI/ISO SQL Standard

Compatible with ANSI/ISO SQL Standard if the scale option is set to zero.

## Related Information

[UltraLite scale Creation Option \[page 169\]](#)

[DECIMAL Data Type \[page 297\]](#)

### 1.13.2.2.8 REAL Data Type

The REAL data type stores single-precision floating-point numbers stored in 4 bytes.

☰ Syntax

*REAL*

## Remarks

The REAL data type is an approximate numeric data type and subject to rounding errors after arithmetic operations. The approximate nature of REAL values means that queries using equalities should generally be avoided when comparing REAL values.

REAL values require 4 bytes of storage.

The range of values is  $-3.402823e+38$  to  $3.402823e+38$ , with numbers close to zero as small as  $1.175494351e-38$ . Values held as REAL are accurate to 7 significant digits, but may be subject to rounding error beyond the sixth digit.

## Standards

**ANSI/ISO SQL Standard**

Core Feature.

## Related Information

[Numeric Functions \[page 332\]](#)

[Aggregate Functions \[page 324\]](#)

[DOUBLE Data Type \[page 298\]](#)

[DOUBLE Data Type \[page 298\]](#)

[DECIMAL Data Type \[page 297\]](#)

[NUMERIC Data Type \[page 301\]](#)

## 1.13.2.2.9 SMALLINT Data Type

The SMALLINT data type stores integers that require 2 bytes of storage.

☰, Syntax

```
[ UNSIGNED ] SMALLINT
```

### Remarks

The SMALLINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations. It requires 2 bytes of storage.

The range for SMALLINT values is  $-2^{15}$  to  $2^{15} - 1$ , or -32768 to 32767.

The range for UNSIGNED SMALLINT values is 0 to  $2^{16} - 1$ , or 0 to 65535.

When converting a string to a SMALLINT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

### Standards

#### ANSI/ISO SQL Standard

Compatible with the standard. However, the UNSIGNED keyword is not in the standard.

#### MySQL

The UNSIGNED keyword may follow SMALLINT.

### Related Information

[Numeric Functions \[page 332\]](#)

[Aggregate Functions \[page 324\]](#)

[BIGINT Data Type \[page 295\]](#)

[BIT Data Type \[page 296\]](#)

[INTEGER Data Type \[page 300\]](#)

[TINYINT Data Type \[page 305\]](#)



## 1.13.2.2.10 TINYINT Data Type

The TINYINT data type stores unsigned integers requiring 1 byte of storage.

☞ Syntax

*TINYINT*

### Remarks

The TINYINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

The range for TINYINT values is 0 to  $2^8 - 1$ , or 0 to 255.

When converting a string to a TINYINT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

In Embedded SQL, TINYINT columns should not be fetched into variables defined as CHAR or UNSIGNED CHAR, since the result is an attempt to convert the value of the column to a string and then assign the first byte to the variable in the program. Instead, TINYINT columns should be fetched into 2-byte or 4-byte integer columns. To send a TINYINT value to a database from an application written in C, the type of the C variable should be INTEGER.

**UltraLite:** In Embedded SQL, TINYINT columns should not be fetched into variables defined as CHAR, since the result is an attempt to convert the value of the column to a string and then assign the first byte to the variable in the program. Instead, TINYINT columns should be fetched into 2-byte or 4-byte integer columns. To send a TINYINT value to a database from an application written in C, the type of the C variable should be INTEGER.

### Standards

#### ANSI/ISO SQL Standard

Not in the standard.

#### MySQL

The UNSIGNED keyword may precede or follow TINYINT, but the UNSIGNED modifier has no effect as the type is always unsigned.

### Related Information

[Numeric Functions \[page 332\]](#)

[Aggregate Functions \[page 324\]](#)

[BIGINT Data Type \[page 295\]](#)

[BIT Data Type \[page 296\]](#)

[VARCHAR Data Type \[page 292\]](#)

[SMALLINT Data Type \[page 304\]](#)

## 1.13.2.3 Date and Time Data Types

Date values can be output in full century format, and the internal storage of dates always explicitly includes the century portion of a year value.

Correct values are always returned for any legal arithmetic and logical operations on dates, regardless of whether the calculated values span different centuries.

### In this section:

[DATE Data Type \[page 306\]](#)

The DATE data type stores calendar dates, such as a year, month, and day.

[DATETIME Data Type \[page 308\]](#)

The DATETIME data type is a Transact-SQL compatible alias for TIMESTAMP, used to store date and time of day.

[DATETIMEOFFSET Data Type \[page 309\]](#)

The DATETIMEOFFSET data type is a Transact-SQL compatible alias for TIMESTAMP WITH TIME ZONE, used to store date, time of day, and time zone information.

[TIME Data Type \[page 310\]](#)

The TIME data type stores the time of day, containing the hour, minute, second, and fraction of a second.

[TIMESTAMP Data Type \[page 312\]](#)

The TIMESTAMP data type stores a point in time containing the year, month, day, hour, minute, second, and fraction of a second stored to 6 decimal places.

[TIMESTAMP WITH TIME ZONE Data Type \[page 313\]](#)

The TIMESTAMP WITH TIME ZONE data type stores a point in time with a time zone offset.

### 1.13.2.3.1 DATE Data Type

The DATE data type stores calendar dates, such as a year, month, and day.

☞ Syntax

```
DATE
```

## Remarks

A DATE value requires 4 bytes of storage.

The format in which DATE values are retrieved as strings by applications is controlled by the `date_format` option setting. For example, a DATE value representing the 19th of July, 2010 can be returned to an application as 2010/07/19, or as Jul 19, 2010 depending on the `date_format` option setting.

**UltraLite:** The format in which DATE values are retrieved as strings by applications is controlled by the `date_format` creation parameter. For example, a DATE value representing the 19th of July, 2010 can be returned to an application as 2010/07/19, or as Jul 19, 2010 depending on the `date_format` creation parameter.

## Standards

### ANSI/ISO SQL Standard

A feature in the standard.

### Transact-SQL

Supported by Adaptive Server Enterprise.

## Related Information

[CURRENT TIME Special Value - UltraLite \[page 260\]](#)  
[CURRENT TIMESTAMP Special Value - UltraLite \[page 261\]](#)  
[Date and Time Functions \[page 326\]](#)  
[CURRENT TIMESTAMP Special Value - UltraLite \[page 261\]](#)  
[DATE Function \[Date and Time\] \[page 376\]](#)  
[UltraLite `date\_format` Creation Option \[page 151\]](#)  
[UltraLite `date\_order` Creation Option \[page 153\]](#)  
[DATETIME Data Type \[page 308\]](#)  
[DATETIME Function \[Date and Time\] \[page 385\]](#)  
[ISDATE Function \[Data Type Conversion\] \[page 412\]](#)  
[UltraLite `nearest\_century` Creation Option \[page 162\]](#)  
[NOW Function \[Date and Time\] \[page 446\]](#)  
[TIME Data Type \[page 310\]](#)  
[TIMESTAMP Data Type \[page 312\]](#)  
[TIMESTAMP WITH TIME ZONE Data Type \[page 313\]](#)

## 1.13.2.3.2 DATETIME Data Type

The DATETIME data type is a Transact-SQL compatible alias for TIMESTAMP, used to store date and time of day.

☞ Syntax

*DATETIME*

### Remarks

A DATETIME value requires 8 bytes of storage.

The format in which DATETIME values are retrieved as strings by applications is controlled by the `timestamp_format` option setting. For example, the DATETIME value `2010/04/01T23:59:59.999999` can be returned to an application as `2010/04/01 23:59:59`, or as `April 1, 2010 23:59:59.999999` depending on the `timestamp_format` option setting.

### i Note

Although the range of possible dates for the DATETIME data type is the same as the DATE type (covering years 0001 to 9999), the useful range of the DATETIME date type is from 0001-01-01 00:00:00 up to, but not including, 7911-01-01 00:00:00. Beyond this range, the hours and minutes portion of the DATETIME value is not retained, but seconds and fractional seconds are. In this case, built-in functions that pertain to minutes or seconds may produce meaningless results.

When a DATETIME value is converted to a DATETIMEOFFSET, the connection's `time_zone_adjustment` setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a DATETIMEOFFSET value is converted to DATETIME, the offset is discarded.

**UltraLite:** When a DATETIME value is converted to DATETIMEOFFSET, the local time zone offset on the system is used in the final result.

### Standards

#### ANSI/ISO SQL Standard

Not in the standard.

#### Transact-SQL

DATETIME, rather than TIMESTAMP, is used by Adaptive Server Enterprise. The DATETIME type in Adaptive Server Enterprise supports dates between January 1, 1753 and December 31, 9999 and supports less precision with the time portion of the value. In SQL Anywhere, DATETIME is implemented as a TIMESTAMP without these restrictions. You should be aware of these differences when migrating data between SQL Anywhere and Adaptive Server Enterprise.

## Related Information

[TIMESTAMP Data Type \[page 312\]](#)

### 1.13.2.3.3 DATETIMEOFFSET Data Type

The DATETIMEOFFSET data type is a Transact-SQL compatible alias for `TIMESTAMP WITH TIME ZONE`, used to store date, time of day, and time zone information.

#### ☰ Syntax

`DATETIMEOFFSET`

## Remarks

The DATETIMEOFFSET value contains the year, month, day, hour, minute, second, fraction of a second, and number of hours and minutes before or after Coordinated Universal Time (UTC). The fraction is stored to 6 decimal places.

A DATETIMEOFFSET value requires 10 bytes of storage.

The format in which DATETIMEOFFSET values are retrieved as strings by applications is controlled by the `timestamp_with_time_zone_format` setting. For example, the DATETIMEOFFSET value `2010/04/01T23:59:59.999999-6:00` can be returned to an application as `2010/04/01 23:59:59 -06:00` or as `April 1, 2010 23:59:59.999999 -06:00`, depending on the `timestamp_with_time_zone_format` setting.

#### i Note

Although the range of possible dates for the DATETIMEOFFSET data type is the same as the DATE type (covering years 0001 to 9999), the useful range of the DATETIMEOFFSET date type is from `0001-01-01 00:00:00` up to, but not including, `7911-01-01 00:00:00`. Beyond this range, the hours and minutes portion of the DATETIMEOFFSET value is not retained, but seconds and fractional seconds are. In this case, built-in functions that pertain to minutes or seconds may produce meaningless results.

Do not use DATETIMEOFFSET for computed columns or in materialized views because the value of the governing `time_zone_adjustment` option varies between connections based on their location and the time of year.

Two DATETIMEOFFSET values are considered identical when they represent the same instant in UTC, regardless of the TIME ZONE offset applied. For example, the following statement returns Yes because the results are considered identical:

```
SELECT
IF CAST('2009-07-15 08:00:00 -08:00' AS DATETIMEOFFSET) =
   CAST('2009-07-15 11:00:00 -05:00' AS DATETIMEOFFSET)
   THEN 'Yes'
   ELSE 'No'
```

```
END IF;
```

If you omit the time zone offset from a DATETIMEOFFSET value, it defaults to the current UTC offset of the client regardless of whether the timestamp represents a date and time in standard time or daylight time. For example, if the client is located in the Eastern Standard time zone and executes the following statement while daylight time is in effect, then a timestamp with a time zone appropriate for the Atlantic Standard time zone (-4 hours from UTC) is returned.

```
SELECT CAST('2009/01/30 12:34:55' AS DATETIMEOFFSET)
```

### Comparing DATETIMEOFFSET with other data types

The comparison of DATETIMEOFFSET values with timestamps without time zones is not recommended because the default time zone offset of the client varies with the geographic location of the client and with the time of the year.

Execute the following statement to determine the current time zone offset in minutes for a client:

```
SELECT CONNECTION_PROPERTY( 'TimeZoneAdjustment' );
```

**UltraLite:** The TimeZoneAdjustment connection property is not supported in UltraLite databases.

### Converting to or from DATETIMEOFFSET

When a DATETIME value is converted to DATETIMEOFFSET, the connection's time\_zone\_adjustment setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a DATETIMEOFFSET value is converted to DATETIME, the offset is discarded. Conversions to or from types other than strings, date, or date-time types is not supported.

**UltraLite:** When a DATETIME value is converted to DATETIMEOFFSET, the client's time zone is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a DATETIMEOFFSET value is converted to DATETIME, the offset is discarded. Conversions to or from types other than strings, date, or date-time types is not supported.

## Standards

### ANSI/ISO SQL Standard

The specific use of DATETIMEOFFSET is not in the standard. To be compatible with the ANSI/ISO SQL Standard, use TIMESTAMP WITH TIME ZONE. The TIMESTAMP WITH TIME ZONE type is optional ANSI/ISO SQL Language Feature F411.

## 1.13.2.3.4 TIME Data Type

The TIME data type stores the time of day, containing the hour, minute, second, and fraction of a second.

☰ Syntax

*TIME*

## Remarks

A TIME value requires 8 bytes of storage.

When using ODBC, a TIME value sent or retrieved as a binary value (using an ODBC TIME\_STRUCT structure) is restricted to an accuracy of hours, minutes, and seconds. Fractional seconds are not part of the structure. For this reason, TIME values should be sent or retrieved as strings if increased accuracy is desired. The format in which TIME values are retrieved as strings by applications is controlled by the time\_format option setting. For example, the TIME value 23:59:59.999999 can be returned to an application as 23:59:59, 23:59:59.999, or 23:59:59.999999 depending on the time\_format option setting.

**UltraLite:** The format in which TIME values are retrieved as strings by applications is controlled by the time\_format creation parameter. For example, the TIME value 23:59:59.999999 can be returned to an application as 23:59:59, 23:59:59.999, or 23:59:59.999999 depending on the time\_format creation parameter.

## Standards

### ANSI/ISO SQL Standard

A feature in the standard.

### Transact-SQL

The TIME data type is supported by Adaptive Server Enterprise. However, Adaptive Server Enterprise supports millisecond resolution (three digits) rather than microsecond resolution (six digits). You should be aware of these differences when migrating data between SQL Anywhere and Adaptive Server Enterprise. To migrate TIME values, use the Adaptive Server Enterprise BIGTIME data type.

## Related Information

- [CURRENT TIME Special Value - UltraLite \[page 260\]](#)
- [CURRENT TIMESTAMP Special Value - UltraLite \[page 261\]](#)
- [CURRENT UTC TIMESTAMP Special Value - UltraLite \[page 262\]](#)
- [Date and Time Functions \[page 326\]](#)
- [DATE Data Type \[page 306\]](#)
- [DATETIME Data Type \[page 308\]](#)
- [DATE Function \[Date and Time\] \[page 376\]](#)
- [DATETIME Function \[Date and Time\] \[page 385\]](#)
- [Expressions in UltraLite \[page 264\]](#)
- [GETDATE Function \[Date and Time\] \[page 401\]](#)
- [ISDATE Function \[Data Type Conversion\] \[page 412\]](#)
- [NOW Function \[Date and Time\] \[page 446\]](#)
- [TIMESTAMP Data Type \[page 312\]](#)
- [TIMESTAMP WITH TIME ZONE Data Type \[page 313\]](#)
- [UltraLite timestamp\\_format Creation Option \[page 173\]](#)

### 1.13.2.3.5 TIMESTAMP Data Type

The TIMESTAMP data type stores a point in time containing the year, month, day, hour, minute, second, and fraction of a second stored to 6 decimal places.

#### ☰ Syntax

*TIMESTAMP*

#### Remarks

A TIMESTAMP value requires 8 bytes of storage.

The format in which TIMESTAMP values are retrieved as strings by applications is controlled by the timestamp\_format setting. For example, the TIMESTAMP value 2010/04/01T23:59:59.999999 can be returned to an application as 2010/04/01 23:59:59 or as April 1, 2010 23:59:59.999999, depending on the timestamp\_format setting.

#### i Note

Although the range of possible dates for the TIMESTAMP data type is the same as the DATE type (covering years 0001 to 9999), the useful range of the TIMESTAMP date type is from 0001-01-01 00:00:00 up to, but not including, 7911-01-01 00:00:00. Beyond this range, the hours and minutes portion of the TIMESTAMP value is not retained, but seconds and fractional seconds are. In this case, built-in functions that pertain to minutes or seconds may produce meaningless results.

When a TIMESTAMP value is converted to TIMESTAMP WITH TIME ZONE, the connection's time\_zone\_adjustment setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a TIMESTAMP WITH TIME ZONE value is converted to TIMESTAMP, the offset is discarded.

**UltraLite:** When a TIMESTAMP value is converted to TIMESTAMP WITH TIME ZONE, the local time zone offset on the system is used in the final result.

#### Standards

##### ANSI/ISO SQL Standard

Compatible with the standard.

##### Transact-SQL

Adaptive Server Enterprise uses the DATETIME type for TIMESTAMP values.



## Related Information

[CURRENT TIME Special Value - UltraLite \[page 260\]](#)  
[CURRENT TIMESTAMP Special Value - UltraLite \[page 261\]](#)  
[CURRENT UTC TIMESTAMP Special Value - UltraLite \[page 262\]](#)  
[Date and Time Functions \[page 326\]](#)  
[DATE Data Type \[page 306\]](#)  
[DATETIME Data Type \[page 308\]](#)  
[DATE Function \[Date and Time\] \[page 376\]](#)  
[DATETIME Function \[Date and Time\] \[page 385\]](#)  
[UltraLite date\\_order Creation Option \[page 153\]](#)  
[Expressions in UltraLite \[page 264\]](#)  
[GETDATE Function \[Date and Time\] \[page 401\]](#)  
[ISDATE Function \[Data Type Conversion\] \[page 412\]](#)  
[UltraLite nearest\\_century Creation Option \[page 162\]](#)  
[NOW Function \[Date and Time\] \[page 446\]](#)  
[TIME Data Type \[page 310\]](#)  
[TIMESTAMP WITH TIME ZONE Data Type \[page 313\]](#)  
[UltraLite timestamp\\_format Creation Option \[page 173\]](#)  
[UltraLite timestamp\\_with\\_time\\_zone\\_format Creation Option \[page 177\]](#)

### 1.13.2.3.6 TIMESTAMP WITH TIME ZONE Data Type

The `TIMESTAMP WITH TIME ZONE` data type stores a point in time with a time zone offset.

☰ Syntax

```
TIMESTAMP WITH TIME ZONE
```

#### Remarks

The `TIMESTAMP WITH TIME ZONE` value contains the year, month, day, hour, minute, second, fraction of a second, and number of hours and minutes before or after Coordinated Universal Time (UTC). The fraction is stored to 6 decimal places.

A `TIMESTAMP WITH TIME ZONE` value requires 10 bytes of storage.

The format in which `TIMESTAMP WITH TIME ZONE` values are retrieved as strings by applications is controlled by the `timestamp_with_time_zone_format` setting. For example, the `TIMESTAMP WITH TIME ZONE` value `2010/04/01T23:59:59.999999-6:00` can be returned to an application as `2010/04/01 23:59:59 -06:00` or as `April 1, 2010 23:59:59.999999 -06:00`, depending on the `timestamp_with_time_zone_format` setting.

## i Note

Although the range of possible dates for the `TIMESTAMP WITH TIME ZONE` data type is the same as the `DATE` type (covering years 0001 to 9999), the useful range of the `TIMESTAMP WITH TIME ZONE` date type is from 0001-01-01 00:00:00 up to, but not including, 7911-01-01 00:00:00. Beyond this range, the hours and minutes portion of the `TIMESTAMP WITH TIME ZONE` value is not retained, but seconds and fractional seconds are. In this case, built-in functions that pertain to minutes or seconds may produce meaningless results.

Do not use `TIMESTAMP WITH TIME ZONE` for computed columns or in materialized views because the value of the governing `time_zone_adjustment` option varies between connections based on their location and the time of year.

Two `TIMESTAMP WITH TIME ZONE` values are considered identical when they represent the same instant in UTC, regardless of the `TIME ZONE` offset applied. For example, the following statement returns Yes because the results are considered identical:

```
SELECT
IF CAST('2009-07-15 08:00:00 -08:00' AS TIMESTAMP WITH TIME ZONE) =
   CAST('2009-07-15 11:00:00 -05:00' AS TIMESTAMP WITH TIME ZONE)
   THEN 'Yes'
   ELSE 'No'
END IF;
```

If you omit the time zone offset from a `TIMESTAMP WITH TIME ZONE` value, it defaults to the current UTC offset of the client regardless of whether the timestamp represents a date and time in standard time or daylight time. For example, if the client is located in the Eastern Standard time zone and executes the following statement while daylight time is in effect, then a timestamp with a time zone appropriate for the Atlantic Standard time zone (-4 hours from UTC) is returned.

```
SELECT CAST('2009/01/30 12:34:55' AS TIMESTAMP WITH TIME ZONE)
```

### Comparing `TIMESTAMP WITH TIME ZONE` with other data types

The comparison of `TIMESTAMP WITH TIME ZONE` values with timestamps without time zones is not recommended because the default time zone offset of the client varies with the geographic location of the client and with the time of the year.

Execute the following statement to determine the current time zone offset in minutes for a client:

```
SELECT CONNECTION_PROPERTY('TimeZoneAdjustment');
```

**UltraLite:** The `TimeZoneAdjustment` connection property is not supported in UltraLite databases.

### Converting to or from `TIMESTAMP WITH TIME ZONE`

When a `TIMESTAMP` value is converted to `TIMESTAMP WITH TIME ZONE`, the connection's `time_zone_adjustment` setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a `TIMESTAMP WITH TIME ZONE` value is converted to `TIMESTAMP`, the offset is discarded. Conversions to or from types other than strings, date, or date-time types is not supported.

**UltraLite:** When a `TIMESTAMP` value is converted to `TIMESTAMP WITH TIME ZONE`, the client's time zone is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a `TIMESTAMP WITH TIME ZONE` value is converted to `TIMESTAMP`, the offset is discarded. Conversions to or from types other than strings, date, or date-time types is not supported.

## Standards

### ANSI/ISO SQL Standard

TIMESTAMP WITH TIME ZONE is part of optional ANSI/ISO SQL Language Feature F411.

## Related Information

[CURRENT TIME Special Value - UltraLite \[page 260\]](#)  
[CURRENT TIMESTAMP Special Value - UltraLite \[page 261\]](#)  
[CURRENT UTC TIMESTAMP Special Value - UltraLite \[page 262\]](#)  
[Date and Time Functions \[page 326\]](#)  
[DATE Data Type \[page 306\]](#)  
[DATETIME Data Type \[page 308\]](#)  
[DATE Function \[Date and Time\] \[page 376\]](#)  
[DATETIME Function \[Date and Time\] \[page 385\]](#)  
[UltraLite date\\_order Creation Option \[page 153\]](#)  
[Expressions in UltraLite \[page 264\]](#)  
[GETDATE Function \[Date and Time\] \[page 401\]](#)  
[ISDATE Function \[Data Type Conversion\] \[page 412\]](#)  
[UltraLite nearest\\_century Creation Option \[page 162\]](#)  
[NOW Function \[Date and Time\] \[page 446\]](#)  
[TIME Data Type \[page 310\]](#)  
[TIMESTAMP Data Type \[page 312\]](#)  
[UltraLite timestamp\\_format Creation Option \[page 173\]](#)  
[UltraLite timestamp\\_with\\_time\\_zone\\_format Creation Option \[page 177\]](#)

### 1.13.2.4 Binary Data Types

Binary data types store binary data, including images and other types of information that are not interpreted by the database.

#### In this section:

[BINARY Data Type \[page 316\]](#)

The BINARY data type stores binary data of a specified maximum length (in bytes).

[LONG BINARY Data Type \[page 317\]](#)

The LONG BINARY data type stores binary data of arbitrary length.

[UNIQUEIDENTIFIER Data Type \[page 318\]](#)

The UNIQUEIDENTIFIER data type stores UUID (also known as GUID) values.

[VARBINARY Data Type \[page 319\]](#)

The VARBINARY data type stores binary data of a specified maximum length (in bytes).

## 1.13.2.4.1 BINARY Data Type

The BINARY data type stores binary data of a specified maximum length (in bytes).

☞ Syntax

```
BINARY [ ( max-length ) ]
```

### Parameters

#### **max-length**

The maximum length of the value, in bytes. If the length is not specified, then it is 1.

The length must be in the 1 to 32767 range.

### Remarks

During comparisons, BINARY values are compared exactly byte for byte. This differs from the CHAR data type, where values are compared using the collation sequence of the database.

If one binary string is a prefix of the other, the shorter string is considered to be less than the longer string.

Unlike CHAR values, BINARY values are not transformed during character set conversion.

BINARY is semantically equivalent to VARBINARY. It is a variable-length type. In other database management systems, BINARY is a fixed-length type.

**UltraLite:** BINARY is a domain, implemented as VARBINARY.

### Standards

#### **ANSI/ISO SQL Standard**

SQL Language Feature T021.

### Related Information

[VARBINARY Data Type \[page 319\]](#)

## 1.13.2.4.2 LONG BINARY Data Type

The LONG BINARY data type stores binary data of arbitrary length.

☞ Syntax

*LONG BINARY*

### Remarks

The maximum size in bytes is 2 GB minus 1 byte ( $2^{31} - 1$ ) or 2 147 483 647.

#### UltraLite:

- You can cast strings to/from LONG BINARY data.
- LONG BINARY data cannot be concatenated.
- LONG BINARY columns can be included in the result set of a SELECT query.
- Indexes cannot be created on a LONG BINARY type.
- A LONG BINARY type can only be used in the LENGTH and CAST functions.
- Conditions in SQL statements, such as in the WHERE clause, cannot operate on LONG BINARY columns.
- Only INSERT, UPDATE, and DELETE operations are allowed on LONG BINARY column.

### Standards

#### ANSI/ISO SQL Standard

The LONG BINARY data type comprises SQL Language Features T021, "BINARY and VARBINARY data types", and T041, "Basic LOB data type support".

### Related Information

[BINARY Data Type \[page 316\]](#)

[VARBINARY Data Type \[page 319\]](#)

### 1.13.2.4.3 UNIQUEIDENTIFIER Data Type

The UNIQUEIDENTIFIER data type stores UUID (also known as GUID) values.

☞ Syntax

*UNIQUEIDENTIFIER*

#### Remarks

The UNIQUEIDENTIFIER data type is typically used for a primary key or other unique column to hold UUID (Universally Unique Identifier) values that uniquely identify rows. The NEWID function generates UUID values in such a way that a value produced on one computer does not match a UUID produced on another computer. UNIQUEIDENTIFIER values generated using NEWID can therefore be used as keys in a synchronization environment.

For example:

```
CREATE TABLE T1 (  
    pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),  
    c1 INT );
```

UUID values are also referred to as GUID (Globally Unique Identifier) values.

UNIQUEIDENTIFIER values are stored as BINARY(16) but are described to client applications as BINARY(36). This description ensures that if the client fetches the value as a string, it has allocated enough space for the result.

For SQL Anywhere ODBC client applications, uniqueidentifier values appear as a SQL\_GUID type.

UNIQUEIDENTIFIER values are automatically converted between string and binary values as needed. Input string values may contain hyphens, but must be properly formatted if they do. The following illustrates two permissible input formats.

```
SELECT STRTOUUID('9752b904beef4bd8adb5642ea2c71986'),  
       STRTOUUID('9752b904-beef-4bd8-adb5-642ea2c71986');
```

UNIQUEIDENTIFIER string values are formatted with hyphens so they are compatible with other RDBMSs.

You can change this by setting the `uuid_has_hyphens` option to Off.

**UltraLite:** There is no comparable setting for UltraLite. UNIQUEIDENTIFIER string values are always formatted with hyphens. Input string values must be properly formatted with hyphens.

#### Standards

##### ANSI/ISO SQL Standard

Not in the standard.

## Related Information

[NEWID Function \[Miscellaneous\] \[page 445\]](#)

[UUIDTOSTR Function \[String\] \[page 509\]](#)

[STRTOUUID Function \[String\] \[page 491\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.2.4.4 VARBINARY Data Type

The VARBINARY data type stores binary data of a specified maximum length (in bytes).

☰ Syntax

```
VARBINARY [ ( max-length ) ]
```

## Parameters

### max-length

The maximum length of the value, in bytes. If the length is not specified, then it is 1.

The length must be in the 1 to 32767 range.

## Remarks

During comparisons, VARBINARY values are compared exactly byte for byte. This behavior differs from the CHAR data type, where values are compared using the collation sequence of the database.

VARBINARY values are not transformed during character set conversion.

VARBINARY can also be specified as BINARY VARYING. Regardless of which syntax is used, the data type is described as VARBINARY. If one binary string is a prefix of the other, the shorter string is considered to be less than the longer string.

**UltraLite:** If one binary string is a prefix of the other, the shorter string is compared to the other as though the shorter string were padded with zeros. When evaluating expressions, the maximum length for a temporary character value is 2048 bytes.

## Standards

### ANSI/ISO SQL Standard

SQL Language Feature TO21, "BINARY and VARBINARY data types".

## Related Information

[Bitwise Operators - UltraLite \[page 287\]](#)

[BINARY Data Type \[page 316\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.3 Spatial Data Types

**Spatial data** is data that describes the position, shape, and orientation of objects in a defined space. UltraLite provides storage and data management features for spatial data, in the form of points, allowing you to store information such as geographic locations and routing information.

Points are defined using a **spatial type**, ST\_GEOMETRY. You use functions and constructors to access and manipulate the spatial data. UltraLite also provides a set of SQL spatial functions designed for compatibility with other products.

A point defines a single location in space. A point geometry does not have length or area. A point always has an X and Y coordinate.

In GIS data, points are typically used to represent locations such as addresses, or geographic features such as a mountain.

#### In this section:

[ST\\_GEOMETRY Data Type - UltraLite \[page 321\]](#)

The ST\_GEOMETRY type is used to store spatial data in the form of points.

## Related Information

[Recommended Reading on Spatial Topics](#)



## 1.13.3.1 ST\_GEOMETRY Data Type - UltraLite

The ST\_GEOMETRY type is used to store spatial data in the form of points.

### Remarks

The ST\_GEOMETRY type is the maximal supertype of the geometry type hierarchy. The ST\_GEOMETRY type supports methods that can be applied to any spatial value. The ST\_GEOMETRY type cannot be instantiated; instead, a subtype should be instantiated. When working with original formats (WKT or WKB), you can use methods such as ST\_PointFromText/ST\_PointFromWKB to instantiate the appropriate concrete type representing the value in the original format.

The ST\_SRID method can be used to retrieve the spatial reference system associated with the value.

Columns of the ST\_GEOMETRY type or any of its subtypes cannot be included in a primary key, unique index, or unique constraint.

### Column and Object Definitions

UltraLite provides a fixed set of three different reference systems that you can attribute to a column during its creation. Individual geometry objects can be associated with any SRID value except the undefined reference system, and can only be stored in a column associated with a matching SRID value or the undefined reference system.

The predefined reference systems are:

#### Undefined or "null" reference system

This is the default reference system if no SRID value is provided. It allows contained geometry values to be in any valid reference system. This reference system allows for catch-all columns that do not enforce any reference system consistency among their geometry objects.

#### Default planar reference system

Defined by specifying a SRID value of 0 during column creation, this column can contain only geometry values associated with this reference system. The values are treated as being in 2D planar space.

#### WGS 84 Geodetic Reference System

Defined by specifying a SRID value of 4326 during column creation, this column can only contain geometry values associated with this reference system. The values are treated as being on the Earth's surface and operations are applied accordingly.

#### **i** Note

A point in SRID 4326 can be stored in a column with the WGS 84 reference system or with the undefined reference system, but not in the default planar system.

No transformations between reference systems are supported.

## Example

The following example illustrates how to create a table with one column associated with the default planar reference system and one with an undefined reference system:

```
CREATE TABLE T1 (  
  V1 INTEGER PRIMARY KEY,  
  V2 ST_GEOMETRY (SRID=0),  
  V3 ST_GEOMETRY  
)
```

The following SQL statement illustrates how to insert data into the T1 table from the previous example:

```
INSERT INTO T1 (V1, V2, V3)  
VALUES (1, ST_POINTFROMTEXT('POINT(10 20)', 0), ST_POINT(5, 6, 2163))
```

## Related Information

- [ST\\_AsBinary Function \[Spatial\] - UltraLite \[page 477\]](#)
- [ST\\_AsText Function \[Spatial\] - UltraLite \[page 478\]](#)
- [ST\\_Distance Function \[Spatial\] - UltraLite \[page 479\]](#)
- [ST\\_Equals Function \[Spatial\] - UltraLite \[page 480\]](#)
- [ST\\_IntersectsRect Function \[Spatial\] - UltraLite \[page 481\]](#)
- [ST\\_Point Function \[Spatial\] - UltraLite \[page 482\]](#)
- [ST\\_PointFromExtText Function \[Spatial\] - UltraLite \[page 483\]](#)
- [ST\\_PointFromText Function \[Spatial\] - UltraLite \[page 484\]](#)
- [ST\\_PointFromWKB Function \[Spatial\] - UltraLite \[page 485\]](#)
- [ST\\_SRID Function \[Spatial\] - UltraLite \[page 486\]](#)
- [ST\\_X Function \[Spatial\] - UltraLite \[page 487\]](#)
- [ST\\_Y Function \[Spatial\] - UltraLite \[page 488\]](#)

## 1.13.4 User-defined Data Types and Their Equivalents

Unlike SQL Anywhere databases, UltraLite does not support user-defined data types.

The following table lists UltraLite data type equivalents to built-in SQL Anywhere aliases:

SQL Anywhere data type	UltraLite equivalent
MONEY	DECIMAL(19,4)
SMALLMONEY	DECIMAL(10,4)
TEXT	LONG VARCHAR
XML	LONG VARCHAR

## Related Information

[LONG VARCHAR Data Type \[page 291\]](#)

[DECIMAL Data Type \[page 297\]](#)

## 1.13.5 SQL Functions

Functions are used to return information from the database. They can be called anywhere an expression is allowed.

Unless otherwise specified in the documentation, NULL is returned for a function if any argument is NULL.

Functions use the same syntax conventions used by SQL statements.

In SQL Anywhere, if an argument is optional, then DEFAULT can be provided as an argument.

### In this section:

[Function Types \[page 323\]](#)

Functions can be grouped according to the type of data they operate on, or the context in which they are used.

[Functions \[page 338\]](#)

Each function is listed, and the function type (numeric, character, and so on) is indicated next to it.

## Related Information

[Syntax Conventions](#)

### 1.13.5.1 Function Types

Functions can be grouped according to the type of data they operate on, or the context in which they are used.

#### **i** Note

Unless otherwise stated, any SQL Anywhere function that receives NULL as a parameter returns NULL.

**UltraLite:** UltraLite supports a subset of the same functions documented for SQL Anywhere, and sometimes with a few differences.

### In this section:

[Aggregate Functions \[page 324\]](#)

Aggregate functions summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement. Aggregate functions are allowed only in the SELECT list and in the HAVING and ORDER BY clauses of a SELECT statement.

#### [Data Type Conversion Functions \[page 326\]](#)

Data type conversion functions are used to convert arguments from one data type to another, or to test whether they can be converted.

#### [Date and Time Functions \[page 326\]](#)

Date and time functions perform operations on DATE, TIME, TIMESTAMP, and TIMESTAMP WITH TIME ZONE data types.

#### [Miscellaneous Functions \[page 330\]](#)

Miscellaneous functions perform operations on arithmetic, string, or date/time expressions, including the return values of other functions.

#### [Numeric Functions \[page 332\]](#)

Numeric functions perform mathematical operations on numerical data types or return numeric information.

#### [Spatial Functions - UltraLite \[page 333\]](#)

Spatial data is data that describes the position, shape, and orientation of objects in a defined space. UltraLite provides storage and data management features for spatial data, in the form of points, allowing you to store information such as geographic locations and routing information.

#### [String Functions \[page 334\]](#)

String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

#### [System Functions \[page 336\]](#)

System functions return system information.

## Related Information

[ST\\_GEOMETRY Data Type - UltraLite \[page 321\]](#)

### 1.13.5.1.1 Aggregate Functions

Aggregate functions summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement. Aggregate functions are allowed only in the SELECT list and in the HAVING and ORDER BY clauses of a SELECT statement.

## List of SQL Anywhere Functions

The following aggregate functions are available:

ARRAY\_AGG function [Aggregate]

AVG function [Aggregate]

BIT\_AND function [Aggregate]

BIT\_OR function [Aggregate]

BIT\_XOR function [Aggregate]  
COVAR\_POP function [Aggregate]  
COVAR\_SAMP function [Aggregate]  
COUNT function [Aggregate]  
COUNT\_BIG function [Aggregate]  
CORR function [Aggregate]  
FIRST\_VALUE function [Aggregate]  
GROUPING function [Aggregate]  
LAST\_VALUE function [Aggregate]  
LIST function [Aggregate]  
MAX function [Aggregate]  
MEDIAN function [Aggregate]  
MIN function [Aggregate]  
REGR\_AVGX function [Aggregate]  
REGR\_AVGY function [Aggregate]  
REGR\_COUNT function [Aggregate]  
REGR\_INTERCEPT function [Aggregate]  
REGR\_R2 function [Aggregate]  
REGR\_SLOPE function [Aggregate]  
REGR\_SXX function [Aggregate]  
REGR\_SXY function [Aggregate]  
REGR\_SYY function [Aggregate]  
SET\_BITS function [Aggregate]  
STDDEV function [Aggregate]  
STDDEV\_POP function [Aggregate]  
STDDEV\_SAMP function [Aggregate]  
SUM function [Aggregate]  
VAR\_POP function [Aggregate]  
VAR\_SAMP function [Aggregate]  
VARIANCE function [Aggregate]  
XMLAGG function [Aggregate]

## List of UltraLite Functions

The following aggregate functions are available:

AVG function [Aggregate]  
COUNT function [Aggregate]  
COUNT\_UPLOAD\_ROWS function [Aggregate]  
LIST function [Aggregate]  
MAX function [Aggregate]  
MIN function [Aggregate]  
SUM function [Aggregate]

## 1.13.5.1.2 Data Type Conversion Functions

Data type conversion functions are used to convert arguments from one data type to another, or to test whether they can be converted.

### List of SQL Anywhere Functions

The following data type conversion functions are available:

BINTOHEX Function [Data Type Conversion]

CAST Function [Data Type Conversion]

CONVERT Function [Data Type Conversion]

HEXTOBIN Function [Data Type Conversion]

HEXTOINT Function [Data Type Conversion]

INTTOHEX Function [Data Type Conversion]

ISDATE Function [Data Type Conversion]

ISNUMERIC Function [Miscellaneous]

TREAT Function [Data Type Conversion]

### List of UltraLite Functions

The following data type conversion functions are available:

CAST Function [Data Type Conversion]

CONVERT Function [Data Type Conversion]

HEXTOINT Function [Data Type Conversion]

INTTOHEX Function [Data Type Conversion]

ISDATE Function [Data Type Conversion]

## 1.13.5.1.3 Date and Time Functions

Date and time functions perform operations on DATE, TIME, TIMESTAMP, and TIMESTAMP WITH TIME ZONE data types.

SQL Anywhere includes compatibility support for Transact-SQL date and time types, including DATETIME and SMALLDATETIME. These Transact-SQL data types are implemented as domains over the native TIMESTAMP data type.

The following date and time functions are available for SQL Anywhere:

DATE function [Date and time]

DATEADD function [Date and time]

DATEDIFF function [Date and time]

DATEFORMAT function [Date and time]  
DATENAME function [Date and time]  
DATEPART function [Date and time]  
DATETIME function [Date and time]  
DAY function [Date and time]  
DAYNAME function [Date and time]  
DAYS function [Date and time]  
DOW function [Date and time]  
GETDATE function [Date and time]  
HOUR function [Date and time]  
HOURS function [Date and time]  
MINUTE function [Date and time]  
MINUTES function [Date and time]  
MONTH function [Date and time]  
MONTHNAME function [Date and time]  
MONTHS function [Date and time]  
NOW function [Date and time]  
QUARTER function [Date and time]  
SECOND function [Date and time]  
SECONDS function [Date and time]  
SWITCHOFFSET function [Date and time]  
SYSDATETIMEOFFSET function [Date and time]  
TODAY function [Date and time]  
TODATETIMEOFFSET function [Date and time]  
WEEKS function [Date and time]  
YEAR function [Date and time]  
YEARS function [Date and time]  
YMD function [Date and time]

The following date and time functions are available for UltraLite:

DATE function [Date and time]  
DATEADD function [Date and time]  
DATEDIFF function [Date and time]  
DATEFORMAT function [Date and time]  
DATENAME function [Date and time]  
DATEPART function [Date and time]  
DATETIME function [Date and time]  
DAY function [Date and time]  
DAYNAME function [Date and time]  
DAYS function [Date and time]  
DOW function [Date and time]  
GETDATE function [Date and time]  
HOUR function [Date and time]  
HOURS function [Date and time]  
MINUTE function [Date and time]

MINUTES function [Date and time]  
 MONTH function [Date and time]  
 MONTHNAME function [Date and time]  
 MONTHS function [Date and time]  
 NOW function [Date and time]  
 QUARTER function [Date and time]  
 SECOND function [Date and time]  
 SECONDS function [Date and time]  
 SWITCHOFFSET function [Date and time]  
 TODAY function [Date and time]  
 TODATETIMEOFFSET function [Date and time]  
 WEEKS function [Date and time]  
 YEAR function [Date and time]  
 YEARS function [Date and time]  
 YMD function [Date and time]

**In this section:**

[Specifying Date Parts \[page 328\]](#)

Many of the date functions use dates built from **date parts**. The following table displays the allowed date part specifiers, their short forms, and the range of values returned by the DATEPART function.

### 1.13.5.1.3.1 Specifying Date Parts

Many of the date functions use dates built from **date parts**. The following table displays the allowed date part specifiers, their short forms, and the range of values returned by the DATEPART function.

Date part	Abbreviation	Values
Year	YY	1-9999
Quarter	QQ	1-4
Month	MM	1-12
Week	WK	1-54. Weeks begin on Sunday. A 54-week year occurs in leap years that start on a Saturday. Week is not subject to the first_day_of_week setting.
Day	DD	1-31
Dayofyear	DY	1-366
Weekday	DW	1-7. Weekday is subject to the first_day_of_week setting. For example, If the first day of week is Monday, then Monday is 1 and Sunday is 7.
Hour	HH	0-23
Minute	MI	0-59



Date part	Abbreviation	Values
Second	SS	0-59
Millisecond	MS	0-999
Microsecond	MCS or US	0-999999
Calyearofweek	CYR	1-9999. The year in which the week begins. The ISO standard first full week of any year always begins on a Monday. The first week of the year can start before, on, or after the first day of the year.  If at least the first 4 days of the year occur in a week, that week is considered to be the first week of the year. Any days of the previous calendar year, that also fall in the first week of the year, are included. Otherwise, the next week is the first full week of the year. In this case, any days of the previous week are part of the last full week of the previous year.
Calweekofyear	CWK	1-53. The week number within the year that contains the specified date.  For more information about the ISO week system and the ISO 8601 date and time standard, see <a href="#">ISO week date</a> .
Caldayofweek	CDW	1-7. (Monday = 1, ..., Sunday = 7)
TZOffset	TZ	-840 to 840

Note that Sunday is the last day of the week in the ISO 8601 calendar, whereas Sunday is considered the first day of the week in some locales (for example, the United States, Canada, and Japan).

Calyearofweek, Calweekofyear, and Caldayofweek conform to ISO 8601 in which weeks start with Monday. The first week of a year is the week that contains the first Thursday of the year (and, hence, always contains 4 January). These values are not affected by the `first_day_of_week` option setting.

The ISO standard numbers each weekday as follows: Monday=1, Tuesday=2, ..., Sunday=7. To calculate the first Monday of the year, the week is split into two groups. The first, major group contains the 4 days Monday to Thursday. The second, minor group contains the 3 days Friday to Sunday.

If the first day of the year falls in the first group (Monday to Thursday), then the majority of the days in the week fall in this year and all days of that week including days that are part of the previous year are considered to belong to the first full week of this year. For example, Monday 2014-12-29 occurs in the first full week of 2015 because 2015-01-01 is a Thursday (the majority of the days in that week are part of 2015). Here Calyearofweek (CYR) for those days is 2015.

January 2015

M	T	W	T	F	S	S	CYR	2015
29	30	31	1	2	3	4	CYR	2015
5	6	7	8	9	10	11	CYR	2015
12	13	14	15	16	17	18	CYR	2015
19	20	21	22	23	24	25	CYR	2015
26	27	28	29	30	31		CYR	2015

If the first day of the year falls in the second group (Friday to Sunday), then the first Monday of the year falls in the next week. In this case, the first few days of the year before that Monday are considered to fall in the last full week of the previous year. For example, Friday 2016-01-01 to Sunday 2016-01-03 fall in the last full week of 2015 (the majority of the days in that week are part of 2015). Here Calyearofweek (CYR) for those days is 2015, not 2016.

January 2016								
M	T	W	T	F	S	S	CYR	2015
28	29	30	31	1	2	3	CYR	2015
4	5	6	7	8	9	10	CYR	2016
11	12	13	14	15	16	17	CYR	2016
18	19	20	21	22	23	24	CYR	2016
25	26	27	28	29	30	31	CYR	2016

## Related Information

[SQL Data Types \[page 288\]](#)

### 1.13.5.1.4 Miscellaneous Functions

Miscellaneous functions perform operations on arithmetic, string, or date/time expressions, including the return values of other functions.

#### List of SQL Anywhere Functions

The following miscellaneous functions are available:

- ARGN Function [Miscellaneous]
- COALESCE Function [Miscellaneous]
- CONFLICT Function [Miscellaneous]
- ERRORMSG Function [Miscellaneous]
- ESTIMATE Function [Miscellaneous]
- ESTIMATE\_SOURCE Function [Miscellaneous]
- EXPERIENCE\_ESTIMATE Function [Miscellaneous]
- EXPLANATION Function [Miscellaneous]
- EXPRTYPE Function [Miscellaneous]
- GET\_IDENTITY Function [Miscellaneous]
- GRAPHICAL\_PLAN Function [Miscellaneous]

GREATER Function [Miscellaneous]  
IDENTITY Function [Miscellaneous]  
IFNULL Function [Miscellaneous]  
INDEX\_ESTIMATE Function [Miscellaneous]  
ISNULL Function [Miscellaneous]  
LESSER Function [Miscellaneous]  
NEWID Function [Miscellaneous]  
NULLIF Function [Miscellaneous]  
NUMBER Function [Miscellaneous]  
PLAN Function [Miscellaneous]  
REWRITE Function [Miscellaneous]  
ROW\_NUMBER Function [Miscellaneous]  
SQLDIALECT Function [Miscellaneous]  
SQLFLAGGER Function [Miscellaneous]  
ERROR\_LINE Function [Miscellaneous]  
TRACEBACK Function [Miscellaneous]  
TRANSACTSQL Function [Miscellaneous]  
VAREXISTS Function [Miscellaneous]  
WATCOMSQL Function [Miscellaneous]

## List of UltraLite Functions

The following miscellaneous functions are available:

ARGN Function [Miscellaneous]  
COALESCE Function [Miscellaneous]  
EXPLANATION Function [Miscellaneous]  
GREATER Function [Miscellaneous]  
IFNULL Function [Miscellaneous]  
ISNULL Function [Miscellaneous]  
LESSER Function [Miscellaneous]  
NEWID Function [Miscellaneous]  
NULLIF Function [Miscellaneous]

## 1.13.5.1.5 Numeric Functions

Numeric functions perform mathematical operations on numerical data types or return numeric information.

### List of SQL Anywhere Functions

The following numeric functions are available:

- ABS function [Numeric]
- ACOS function [Numeric]
- ASIN function [Numeric]
- ATAN function [Numeric]
- ATAN2 function [Numeric]
- CEILING function [Numeric]
- COS function [Numeric]
- COT function [Numeric]
- DEGREES function [Numeric]
- EXP function [Numeric]
- FLOOR function [Numeric]
- LOG function [Numeric]
- LOG10 function [Numeric]
- MOD function [Numeric]
- PI function [Numeric]
- POWER function [Numeric]
- RADIANS function [Numeric]
- RAND function [Numeric]
- REMAINDER function [Numeric]
- ROUND function [Numeric]
- SIGN function [Numeric]
- SIN function [Numeric]
- SQRT function [Numeric]
- TAN function [Numeric]
- TRUNCNUM function [Numeric]

### List of UltraLite Functions

The following numeric functions are available:

- ABS function [Numeric]
- ACOS function [Numeric]
- ASIN function [Numeric]
- ATAN function [Numeric]

ATAN2 function [Numeric]  
CEILING function [Numeric]  
COS function [Numeric]  
COT function [Numeric]  
DEGREES function [Numeric]  
EXP function [Numeric]  
FLOOR function [Numeric]  
LOG function [Numeric]  
LOG10 function [Numeric]  
MOD function [Numeric]  
PI function [Numeric]  
POWER function [Numeric]  
RADIANS function [Numeric]  
REMAINDER function [Numeric]  
ROUND function [Numeric]  
SIGN function [Numeric]  
SIN function [Numeric]  
SQRT function [Numeric]  
TAN function [Numeric]  
TRUNCNUM function [Numeric]

### 1.13.5.1.6 Spatial Functions - UltraLite

Spatial data is data that describes the position, shape, and orientation of objects in a defined space. UltraLite provides storage and data management features for spatial data, in the form of points, allowing you to store information such as geographic locations and routing information.

UltraLite provides a set of SQL spatial functions designed for compatibility with other products. You use these functions and constructors to access and manipulate the spatial data.

#### List of Functions

The following spatial functions are available:

- ST\_AsBinary function [Spatial] - UltraLite
- ST\_AsExtText function [Spatial] - UltraLite
- ST\_AsText function [Spatial] - UltraLite
- ST\_Distance function [Spatial] - UltraLite
- ST\_Equals function [Spatial] - UltraLite
- ST\_IntersectsRect function [Spatial] - UltraLite
- ST\_Point function [Spatial] - UltraLite
- ST\_PointFromExtText function [Spatial] - UltraLite

- ST\_PointFromText function [Spatial] - UltraLite
- ST\_PointFromWKB function [Spatial] - UltraLite
- ST\_SRID function [Spatial] - UltraLite
- ST\_X function [Spatial] - UltraLite
- ST\_Y function [Spatial] - UltraLite

### 1.13.5.17 String Functions

String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

When working in a multibyte character set, check carefully whether the function being used returns information concerning characters or bytes.

#### List of SQL Anywhere Functions

The following string functions are available:

ASCII function [String]  
 BASE64\_DECODE function [String]  
 BASE64\_ENCODE function [String]  
 BYTE\_LENGTH function [String]  
 BYTE\_SUBSTR function [String]  
 CHAR function [String]  
 CHARINDEX function [String]  
 CHAR\_LENGTH function [String]  
 COMPARE function [String]  
 COMPRESS function [String]  
 CSCONVERT function [String]  
 DECOMPRESS function [String]  
 DECRYPT function [String]  
 DIFFERENCE function [String]  
 ENCRYPT function [String]  
 HASH function [String]  
 INSERTSTR function [String]  
 LCASE function [String]  
 LEFT function [String]  
 LENGTH function [String]  
 LOCATE function [String]  
 LOWER function [String]  
 LTRIM function [String]  
 NCHAR function [String]

PATINDEX function [String]  
READ\_CLIENT\_FILE function [String]  
READ\_SERVER\_FILE function [String]  
REGEXP\_SUBSTR function [String]  
REPEAT function [String]  
REPLACE function [String]  
REPLICATE function [String]  
REVERSE function [String]  
RIGHT function [String]  
RTRIM function [String]  
SIMILAR function [String]  
SORTKEY function [String]  
SOUNDEX function [String]  
SPACE function [String]  
STR function [String]  
STRING function [String]  
STRTOUUID function [String]  
STUFF function [String]  
SUBSTRING function [String]  
TO\_CHAR function [String]  
TO\_NCHAR function [String]  
TRIM function [String]  
UCASE function [String]  
UNICODE function [String]  
UNISTR function [String]  
UPPER function [String]  
UUIDTOSTR function [String]  
XMLCONCAT function [String]  
XMLELEMENT function [String]  
XMLFOREST function [String]  
XMLGEN function [String]

## List of UltraLite Functions

The following string functions are available:

ASCII function [String]  
BYTE\_LENGTH function [String]  
BYTE\_SUBSTR function [String]  
CHAR function [String]  
CHARINDEX function [String]  
CHAR\_LENGTH function [String]  
DIFFERENCE function [String]  
INSERTSTR function [String]

LCASE function [String]  
LEFT function [String]  
LENGTH function [String]  
LOCATE function [String]  
LOWER function [String]  
LTRIM function [String]  
PATINDEX function [String]  
REPEAT function [String]  
REPLACE function [String]  
REPLICATE function [String]  
RIGHT function [String]  
RTRIM function [String]  
SIMILAR function [String]  
SOUNDEX function [String]  
SPACE function [String]  
STR function [String]  
STRING function [String]  
STRTOUUID function [String]  
STUFF function [String]  
SUBSTRING function [String]  
TRIM function [String]  
UCASE function [String]  
UPPER function [String]  
UUIDTOSTR function [String]

### 1.13.5.1.8 System Functions

System functions return system information.

#### List of Functions

The following system functions are available:

CONNECTION\_EXTENDED\_PROPERTY function [String]  
CONNECTION\_PROPERTY function [System]  
DATALENGTH function [System]  
DB\_ID function [System]  
DB\_NAME function [System]  
DB\_EXTENDED\_PROPERTY function [System]  
DB\_PROPERTY function [System]  
EVENT\_CONDITION function [System]  
EVENT\_CONDITION\_NAME function [System]



EVENT\_PARAMETER function [System]  
NEXT\_CONNECTION function [System]  
NEXT\_DATABASE function [System]  
PROPERTY function [System]  
PROPERTY\_DESCRIPTION function [System]  
PROPERTY\_NAME function [System]  
PROPERTY\_NUMBER function [System]  
SUSER\_ID function [System]  
SUSER\_NAME function [System]  
TSEQUAL function [System] (deprecated)  
USER\_ID function [System]  
USER\_NAME function [System]  
DB\_PROPERTY function [System]

## UltraLite Functions

DB\_PROPERTY function [System]  
ML\_GET\_SERVER\_NOTIFICATION function [System]  
SYNC\_PROFILE\_OPTION\_VALUE function [System]

## SQL Anywhere notes

- The db\_id, db\_name, and datalength functions are implemented as built-in functions.
- Some system functions are implemented as stored procedures.

System functions that are not described elsewhere are noted in the following table. These functions are implemented as stored procedures.

**Syntax:** COL\_LENGTH

```
COL_LENGTH( @object_name, @column_name )
```

Returns the INTEGER defined length of the specified column. @object\_name can contain the owner, for example, 'GROUPO.Customers'.

**Syntax:** COL\_TERM

```
COL_NAME( @object_id, @column_id [, @database_id ] )
```

Returns the CHAR(128) column name.

**Syntax:** INDEX\_COL

```
INDEX_COL ( @table_name, @index_id, @key_# [, @user_id ] )
```

Returns the CHAR(128) name of the indexed column. @table\_name can contain the owner, for example, 'GROUPO.Customers'.

**Syntax:** OBJECT\_ID

```
OBJECT_ID( @object_name )
```

Returns the INTEGER object ID. @object\_name can contain the owner, for example, 'GROUPO.Customers'.

**Syntax:** OBJECT\_NAME

```
OBJECT_NAME ( @object_id [, @database_id ] )
```

Returns the CHAR(128) object name.

## 1.13.5.2 Functions

Each function is listed, and the function type (numeric, character, and so on) is indicated next to it.

**In this section:**

[ABS Function \[Numeric\] \[page 345\]](#)

Returns the absolute value of a numeric expression.

[ACOS Function \[Numeric\] \[page 346\]](#)

Returns the arc-cosine, in radians, of a numeric expression.

[ARGN Function \[Miscellaneous\] \[page 347\]](#)

Returns a selected argument from a list of arguments.

[ASCII Function \[String\] \[page 348\]](#)

Returns the integer ASCII value of the first byte in a string-expression.

[ASIN Function \[Numeric\] \[page 349\]](#)

Returns the arc-sine, in radians, of a number.

[ATAN Function \[Numeric\] \[page 350\]](#)

Returns the arc-tangent, in radians, of a number.

[ATAN2 Function \[Numeric\] \[page 352\]](#)

Returns the arc-tangent, in radians, of the ratio of two numbers.

[AVG Function \[Aggregate\] \[page 353\]](#)

Computes the average, for a set of rows, of a numeric expression or of a set of unique values.

[BYTE\\_LENGTH Function \[String\] \[page 355\]](#)

Returns the number of bytes in a string.

[BYTE\\_SUBSTR Function \[String\] \[page 356\]](#)

Returns a substring of a string. The substring is determined using bytes, not characters.

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

Returns the value of an expression converted to a supplied data type.

[CEILING Function \[Numeric\] \[page 361\]](#)

Returns the first integer that is greater or equal to a given value. For positive numbers, this is known as rounding up.

[CHAR Function \[String\] \[page 362\]](#)

Returns the character with the ASCII value of a number.

[CHAR\\_LENGTH Function \[String\] \[page 363\]](#)

Returns the number of characters in a string.

[CHARINDEX Function \[String\] \[page 364\]](#)

Returns the position of one string in another.

[COALESCE Function \[Miscellaneous\] \[page 366\]](#)

Returns the first non-NULL expression from a list. This function is identical to the ISNULL function.

[CONVERT Function \[Data Type Conversion\] \[page 367\]](#)

Returns an expression converted to a supplied data type.

[COS Function \[Numeric\] \[page 370\]](#)

Returns the cosine of the angle in radians given by its argument.

[COT Function \[Numeric\] \[page 371\]](#)

Returns the cotangent of the angle in radians given by its argument.

[COUNT Function \[Aggregate\] \[page 372\]](#)

Counts the number of rows in a group depending on the specified parameters.

[COUNT\\_UPLOAD\\_ROWS function \[Aggregate\] \[page 374\]](#)

Returns a count of the number of rows that will be uploaded in the next synchronization.

[DATALENGTH Function \[System\] \[page 375\]](#)

Returns the length, in bytes, of the underlying storage for the result of an expression.

[DATE Function \[Date and Time\] \[page 376\]](#)

Converts the expression into a date, and removes any hours, minutes, or seconds.

[DATEADD Function \[Date and Time\] \[page 377\]](#)

Returns a `TIMESTAMP` or `TIMESTAMP WITH TIME ZONE` value produced by adding a date part to its argument.

[DATEDIFF Function \[Date and Time\] \[page 379\]](#)

Returns the interval between two dates.

[DATEFORMAT Function \[Date and Time\] \[page 381\]](#)

Returns a string representing a date expression in the specified format.

[DATENAME Function \[Date and Time\] \[page 382\]](#)

Returns the name of the specified part (such as the month June) of a `TIMESTAMP` or `TIMESTAMP WITH TIME ZONE` value, as a character string.

[DATEPART Function \[Date and Time\] \[page 384\]](#)

Returns a portion of a `TIMESTAMP` or `TIMESTAMP WITH TIME ZONE` value.

[DATETIME Function \[Date and Time\] \[page 385\]](#)

Converts an expression into a `TIMESTAMP` value.

[DAY Function \[Date and Time\] \[page 386\]](#)

Returns the day of the month of its argument as an integer between 1 and 31.

[DAYNAME Function \[Date and Time\] \[page 387\]](#)

Returns the name of the day of the week from a date.

[DAYS Function \[Date and Time\] \[page 388\]](#)

Manipulates a `TIMESTAMP` or returns the number of days between two `TIMESTAMP` values.

[DB\\_PROPERTY Function \[System\] \[page 390\]](#)

Returns the value of the specified database property.

[DEGREES Function \[Numeric\] \[page 392\]](#)

Converts a number from radians to degrees.

[DIFFERENCE Function \[String\] \[page 393\]](#)

Returns the difference in the SOUNDEX values between the two string expressions.

[DOW Function \[Date and Time\] \[page 394\]](#)

Returns a number from 1 to 7 representing the day of the week of a date, where Sunday=1, Monday=2, and so on.

[EXP Function \[Numeric\] \[page 395\]](#)

Returns the result of the base of natural logarithms e raised to the power of the given argument.

[EXPLANATION Function \[Miscellaneous\] \[page 396\]](#)

Returns the optimization strategy of a SQL statement as a plain text string.

[EXTRACT Function \[Date and Time\] \[page 398\]](#)

Returns a date part from a DATE, TIME, TIMESTAMP, or TIMESTAMP WITH TIME ZONE expression.

[FLOOR Function \[Numeric\] \[page 400\]](#)

Returns the largest integer not greater than the given number.

[GETDATE Function \[Date and Time\] \[page 401\]](#)

Returns the current year, month, day, hour, minute, second, and fraction of a second.

[GREATER Function \[Miscellaneous\] \[page 402\]](#)

Returns the greater of two parameter values.

[HEXTOINT Function \[Data Type Conversion\] \[page 403\]](#)

Returns the decimal integer equivalent of a hexadecimal string.

[HOUR Function \[Date and Time\] \[page 405\]](#)

Returns the hour component of a TIMESTAMP value.

[HOURS Function \[Date and Time\] \[page 406\]](#)

Manipulates a TIMESTAMP or returns the number of hours between two TIMESTAMP values.

[IFNULL Function \[Miscellaneous\] \[page 408\]](#)

Evaluates whether one expression is NULL and returns a value.

[INSERTSTR Function \[String\] \[page 409\]](#)

Inserts a string into another string at a specified position.

[INTTOHEX Function \[Data Type Conversion\] \[page 410\]](#)

Returns a string containing the hexadecimal equivalent of an integer.

[ISDATE Function \[Data Type Conversion\] \[page 412\]](#)

Tests if a string argument can be converted to a date.

[ISNULL Function \[Miscellaneous\] \[page 413\]](#)

Returns the first non-NULL expression from a list. This function is identical to the COALESCE function.

[LCASE Function \[String\] \[page 414\]](#)

Converts all characters in a string to lowercase.

[LEFT Function \[String\] \[page 415\]](#)

Returns multiple characters from the beginning of a string.

[LENGTH Function \[String\] \[page 417\]](#)

Returns the number of characters in the specified string.

[LESSER Function \[Miscellaneous\] \[page 418\]](#)

Returns the lesser of two parameter values.

[LIST Function \[Aggregate\] \[page 419\]](#)

Returns a delimited list of values for every row in a group.

[LOCATE Function \[String\] \[page 422\]](#)

Returns the position of one string within another.

[LOG Function \[Numeric\] \[page 424\]](#)

Returns the natural logarithm of a number.

[LOG10 Function \[Numeric\] \[page 425\]](#)

Returns the base 10 logarithm of a number.

[LOWER Function \[String\] \[page 427\]](#)

Converts all characters in a string to lowercase.

[LTRIM Function \[String\] \[page 428\]](#)

Removes leading blanks or specified characters from the string.

[MAX Function \[Aggregate\] \[page 429\]](#)

Returns the maximum expression value found in each group of rows.

[MICROSECOND Function \[Date and Time\] \[page 431\]](#)

Returns the microsecond component of a `TIMESTAMP` expression.

[MILLISECOND Function \[Date and Time\] \[page 432\]](#)

Returns the millisecond component of a `TIMESTAMP` expression.

[MIN Function \[Aggregate\] \[page 433\]](#)

Returns the minimum expression value found in each group of rows.

[MINUTE Function \[Date and Time\] \[page 435\]](#)

Returns the minute component of a `TIMESTAMP` value.

[MINUTES Function \[Date and Time\] \[page 436\]](#)

Manipulates a `TIMESTAMP` or returns the number of minute boundaries between two `TIMESTAMP` values.

[ML\\_GET\\_SERVER\\_NOTIFICATION function \[System\] \[page 438\]](#)

This function allows UltraLite users to use lightweight polling to query a notifier on a MobiLink server for server-initiated sync requests.

[MOD Function \[Numeric\] \[page 439\]](#)

Returns the remainder when one whole number is divided by another.

[MONTH Function \[Date and Time\] \[page 441\]](#)

Returns the month of the given date.

[MONTHNAME Function \[Date and Time\] \[page 442\]](#)

Returns the name of the month from a date.

[MONTHS Function \[Date and Time\] \[page 443\]](#)

Manipulates a `TIMESTAMP` or returns the number of month boundaries between two `TIMESTAMP` values.

[NEWID Function \[Miscellaneous\] \[page 445\]](#)

Generates a UUID (Universally Unique Identifier) value.

[NOW Function \[Date and Time\] \[page 446\]](#)

Returns the current date and time as a `TIMESTAMP` value. The accuracy is limited by the accuracy of the system clock.

[NULLIF Function \[Miscellaneous\] \[page 448\]](#)

Provides an abbreviated CASE expression by comparing expressions.

[PATINDEX Function \[String\] \[page 449\]](#)

Returns an integer representing the starting position of the first occurrence of a pattern in a string.

[PI Function \[Numeric\] \[page 452\]](#)

Returns the numeric value PI.

[POWER Function \[Numeric\] \[page 452\]](#)

Calculates one number raised to the power of another.

[QUARTER Function \[Date and Time\] \[page 453\]](#)

Returns a number indicating the quarter of the year from the supplied `TIMESTAMP` expression.

[RADIANS Function \[Numeric\] \[page 455\]](#)

Converts a number from degrees to radians.

[REMAINDER Function \[Numeric\] \[page 456\]](#)

Returns the remainder when one whole number is divided by another.

[REPEAT Function \[String\] \[page 457\]](#)

Concatenates a string a specified number of times.

[REPLACE Function \[String\] \[page 458\]](#)

Replaces a string with another string, and returns the new results.

[REPLICATE Function \[String\] \[page 460\]](#)

Concatenates a string a specified number of times.

[RIGHT Function \[String\] \[page 461\]](#)

Returns the rightmost characters of a string.

[ROUND Function \[Numeric\] \[page 463\]](#)

Rounds the `numeric-expression` to the specified `integer-expression` amount of places after the decimal point.

[RTRIM Function \[String\] \[page 464\]](#)

Removes trailing blanks or specified characters from the string.

[SECOND Function \[Date and Time\] \[page 466\]](#)

Returns the seconds value of the `TIMESTAMP` argument.

[SECONDS Function \[Date and Time\] \[page 467\]](#)

Manipulates a `TIMESTAMP` or returns the number of second boundaries between two `TIMESTAMP` values.

[SHORT\\_PLAN function \[Miscellaneous\] \[page 469\]](#)

Returns a short description of the UltraLite plan optimization strategy of a SQL statement, as a string. The description is the same as that returned by the `EXPLANATION` function.

[SIGN Function \[Numeric\] \[page 470\]](#)

Returns the sign (positive or negative) of the given number.

[SIMILAR Function \[String\] \[page 471\]](#)

Returns a number indicating the similarity between two strings.

[SIN Function \[Numeric\] \[page 472\]](#)

Returns the sine of a number.

[SOUNDEX Function \[String\] \[page 473\]](#)

Returns a number representing the sound of a string.

[SPACE Function \[String\] \[page 475\]](#)

Returns a specified number of spaces.

[SQRT Function \[Numeric\] \[page 476\]](#)

Returns the square root of a number.

[ST\\_AsBinary Function \[Spatial\] - UltraLite \[page 477\]](#)

Returns a binary string representing the specified geometry.

[ST\\_AsExtText Function \[Spatial\] - UltraLite \[page 478\]](#)

Returns a binary string representing the specified geometry.

[ST\\_AsText Function \[Spatial\] - UltraLite \[page 478\]](#)

Returns a binary string representing the specified geometry.

[ST\\_Distance Function \[Spatial\] - UltraLite \[page 479\]](#)

Returns the smallest distance between two specified geometry values.

[ST\\_Equals Function \[Spatial\] - UltraLite \[page 480\]](#)

Tests whether an ST\_Geometry value is spatially equal to another ST\_Geometry value. Two geometry values can be considered equal if they have the same x and y coordinates and are in the same reference system.

[ST\\_IntersectsRect Function \[Spatial\] - UltraLite \[page 481\]](#)

Tests if a point is located within the box defined by the two points specified as min and max.

[ST\\_Point Function \[Spatial\] - UltraLite \[page 482\]](#)

Constructs a point based on x and y coordinates.

[ST\\_PointFromExtText Function \[Spatial\] - UltraLite \[page 483\]](#)

Returns an ST\_Geometry value, which is transformed from a VARCHAR value containing the EWKT representation of an ST\_Geometry.

[ST\\_PointFromText Function \[Spatial\] - UltraLite \[page 484\]](#)

Returns an ST\_Geometry value, which is transformed from a VARCHAR value containing the WKT representation of an ST\_Geometry.

[ST\\_PointFromWKB Function \[Spatial\] - UltraLite \[page 485\]](#)

Returns an ST\_Geometry value, which is transformed from a BINARY value containing the WKB representation of an ST\_Geometry.

[ST\\_SRID Function \[Spatial\] - UltraLite \[page 486\]](#)

Retrieves the spatial reference system (SRID) associated with the geometry value.

[ST\\_X Function \[Spatial\] - UltraLite \[page 487\]](#)

Returns the x coordinate of the ST\_Geometry value.

[ST\\_Y Function \[Spatial\] - UltraLite \[page 488\]](#)

Returns the y coordinate of the ST\_Geometry value.

[STR Function \[String\] \[page 489\]](#)

Returns the string equivalent of a number.

[STRING Function \[String\] \[page 490\]](#)

Concatenates one or more strings into one large string.

[STRTOUUID Function \[String\] \[page 491\]](#)

Converts a string value to a unique identifier (UUID or GUID) value.

[STUFF Function \[String\] \[page 493\]](#)

Deletes multiple characters from one string and replaces them with another string.

[SUBSTRING Function \[String\] \[page 494\]](#)

Returns a substring of a string.

[SUM Function \[Aggregate\] \[page 497\]](#)

Returns the total of the specified expression for each group of rows.

[SWITCHOFFSET Function \[Date and Time\] \[page 498\]](#)

Returns a `TIMESTAMP WITH TIME ZONE` value that is converted from its original time zone offset to the specified time zone offset.

[SYNC\\_PROFILE\\_OPTION\\_VALUE Function \[System\] - UltraLite \[page 499\]](#)

Returns the value of the option corresponding to the given option name.

[TAN Function \[Numeric\] \[page 500\]](#)

Returns the tangent of a number.

[TODATETIMEOFFSET Function \[Date and Time\] \[page 502\]](#)

Converts a `TIMESTAMP` value to a `TIME STAMP WITH TIME ZONE` value using the specified time zone offset.

[TODAY Function \[Date and Time\] \[page 503\]](#)

Returns the current date as a `DATE` value.

[TRIM Function \[String\] \[page 504\]](#)

Removes leading and trailing blanks or specified characters from a string.

[TRUNCNUM Function \[Numeric\] \[page 505\]](#)

Truncates a number at a specified number of places after the decimal point.

[UCASE Function \[String\] \[page 506\]](#)

Converts all characters in a string to uppercase.

[UPPER Function \[String\] \[page 508\]](#)

Converts all characters in a string to uppercase.

[UUIDTOSTR Function \[String\] \[page 509\]](#)

Converts a unique identifier value (UUID, also known as GUID) to a string value.

[WEEKS Function \[Date and Time\] \[page 511\]](#)

Manipulates a `TIMESTAMP` or returns the number of weeks between two `TIMESTAMP` values.

[YEAR Function \[Date and Time\] \[page 512\]](#)

Returns the year component of the `TIMESTAMP` argument.

[YEARS Function \[Date and Time\] \[page 513\]](#)

Manipulates a `TIMESTAMP` or returns the number of years between two `TIMESTAMP` values.

[YMD Function \[Date and Time\] \[page 515\]](#)

Returns a date value corresponding to the given year, month, and day of the month. Arguments are `INTEGER` values from -32768 to 32767.



## 1.13.5.2.1 ABS Function [Numeric]

Returns the absolute value of a numeric expression.

≡ Syntax

```
ABS( numeric-expression )
```

### Parameters

#### **numeric-expression**

The number whose absolute value is to be returned.

### Returns

An absolute value of the numeric expression.

<b>Numeric-expression data type</b>	<b>Returns</b>
INT	INT
FLOAT	FLOAT
DOUBLE	DOUBLE
NUMERIC	NUMERIC

### Standards

#### **ANSI/ISO SQL Standard**

Part of optional Language Feature T441.

### Example

The following statement returns the value 66:

```
SELECT ABS ( -66 ) ;
```

## 1.13.5.2.2 ACOS Function [Numeric]

Returns the arc-cosine, in radians, of a numeric expression.

☰ Syntax

```
ACOS( numeric-expression )
```

### Parameters

**numeric-expression**

The cosine of the angle.

### Returns

DOUBLE

### Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

### Standards

**ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following statement returns the arc-cosine value for 0.52:

```
SELECT ACOS ( 0.52 );
```

## Related Information

[ASIN Function \[Numeric\] \[page 349\]](#)

[ATAN Function \[Numeric\] \[page 350\]](#)

[ATAN2 Function \[Numeric\] \[page 352\]](#)

[COS Function \[Numeric\] \[page 370\]](#)

### 1.13.5.2.3 ARGN Function [Miscellaneous]

Returns a selected argument from a list of arguments.

☞ Syntax

```
ARGN( integer-expression , expression [ , ... ] )
```

## Parameters

### **integer-expression**

The position of an argument within the list of expressions.

### **expression**

An expression of any data type passed into the function. All supplied expressions must be of the same data type.

## Returns

Using the value of the *integer-expression* as n, returns the nth argument (starting at 1) from the remaining list of arguments.

## Remarks

While the expressions can be of any data type, they must all be of the same data type. The integer expression must be from one to the number of expressions in the list or NULL is returned. Multiple expressions are separated by a comma.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 6:

```
SELECT ARGN ( 6, 1, 2, 3, 4, 5, 6 );
```

## 1.13.5.2.4 ASCII Function [String]

Returns the integer ASCII value of the first byte in a string-expression.

☞ Syntax

```
ASCII( string-expression )
```

## Parameters

**string-expression**

The string.

## Returns

SMALLINT

## Remarks

If the string is empty, then ASCII returns zero. Literal strings must be enclosed in quotes. If the database character set is multibyte and the first character of the parameter string consists of more than one byte, the result is NULL.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 90:

```
SELECT ASCII ( 'Z' );
```

## Related Information

[CHAR Function \[String\] \[page 362\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.5 ASIN Function [Numeric]

Returns the arc-sine, in radians, of a number.

### ☞ Syntax

```
ASIN( numeric-expression )
```

## Parameters

### numeric-expression

The sine of the angle.

## Returns

DOUBLE

## Remarks

The SIN and ASIN functions are inverse operations.

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the arc-sine value for 0.52:

```
SELECT ASIN( 0.52 );
```

## Related Information

[ACOS Function \[Numeric\] \[page 346\]](#)

[ATAN Function \[Numeric\] \[page 350\]](#)

[ATAN2 Function \[Numeric\] \[page 352\]](#)

[SIN Function \[Numeric\] \[page 472\]](#)

## 1.13.5.2.6 ATAN Function [Numeric]

Returns the arc-tangent, in radians, of a number.

☞ Syntax

```
ATAN( numeric-expression )
```

## Parameters

numeric-expression

The tangent of the angle.

## Returns

DOUBLE

## Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

The ATAN and TAN functions are inverse operations.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the arc-tangent value for 0.52:

```
SELECT ATAN( 0.52 );
```

## Related Information

[ACOS Function \[Numeric\] \[page 346\]](#)

[ASIN Function \[Numeric\] \[page 349\]](#)

[ATAN2 Function \[Numeric\] \[page 352\]](#)

[TAN Function \[Numeric\] \[page 500\]](#)

## 1.13.5.2.7 ATAN2 Function [Numeric]

Returns the arc-tangent, in radians, of the ratio of two numbers.

☞ Syntax

```
{ ATAN2 | ATN2 } ( numeric-expression-1 , numeric-expression-2 )
```

### Parameters

#### **numeric-expression-1**

The numerator in the ratio whose arc-tangent is calculated.

#### **numeric-expression-2**

The denominator in the ratio whose arc-tangent is calculated.

### Returns

DOUBLE

### Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

UltraLite does not support the function name short form *ATN2*.

### Standards

#### **ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following statement returns the arc-tangent value for the ratio 0.52 to 0.60:

```
SELECT ATAN2 ( 0.52, 0.60 );
```



## Related Information

[ACOS Function \[Numeric\] \[page 346\]](#)

[ASIN Function \[Numeric\] \[page 349\]](#)

[ATAN Function \[Numeric\] \[page 350\]](#)

[TAN Function \[Numeric\] \[page 500\]](#)

### 1.13.5.2.8 AVG Function [Aggregate]

Computes the average, for a set of rows, of a numeric expression or of a set of unique values.

#### Syntax

##### Numeric expressions

```
AVG( [ ALL | DISTINCT ] numeric-expression )
```

##### Window function

```
AVG( [ ALL ] numeric-expression)OVER( window-spec )
```

*window-spec* : see the Remarks section below

##### UltraLite - numeric expressions

```
AVG( [ DISTINCT ] numeric-expression )
```

## Parameters

### [ ALL ] numeric-expression

The expression whose average is calculated over the rows in each group.

### DISTINCT clause

Computes the average of the unique numeric values in each group.

## Returns

Returns the NULL value for a group containing no rows.

Returns DOUBLE if the argument is DOUBLE, otherwise NUMERIC.

## Remarks

This average does not include rows where the `numeric-expression` is the NULL value.

This function can generate an overflow error, resulting in an error being returned. You can use the CAST function on `numeric-expression` to avoid the overflow error.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

## Standards

### ANSI/ISO SQL Standard

Core Feature. The `numeric-expression` syntax is a Core Feature of the Standard, while `window-spec` syntax comprises part of optional Language Feature T611, "Basic OLAP operations". The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional Language feature F561, "Full value expressions". The software also supports Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references. The software does not support optional Language Feature F442, "Mixed column references in set functions", and it also does not permit the arguments of an aggregate function to include both a column reference from the query block containing the AVG function, combined with an outer reference.

## Example

The following statement returns the value 49988.623200 when connected to the SQL Anywhere 17 Demo:

```
SELECT AVG( Salary ) FROM Employees;
```

The following statement returns the average product price from the Products table when connected to the SQL Anywhere 17 Demo database:

```
SELECT AVG( DISTINCT UnitPrice ) FROM Products;
```

The following statement returns an error with SQLSTATE 42W68 because the arguments of AVG contain both a quantified expression from the subquery, and an outer reference (p.Quantity) from the outer SELECT block when connected to the SQL Anywhere 17 Demo:

```
SELECT * from GROUPO.Products as p
WHERE p.Quantity > ( SELECT AVG( 0.5 * p.Quantity + 0.5 * s.Quantity )
                    from GROUPO.SalesOrderItems as s
                    WHERE s.ProductID = p.ProductID )
```

## Related Information

[SUM Function \[Aggregate\] \[page 497\]](#)

[COUNT Function \[Aggregate\] \[page 372\]](#)

[Troubleshooting Database Upgrades: Aggregate Functions and Outer References](#)

### 1.13.5.2.9 BYTE\_LENGTH Function [String]

Returns the number of bytes in a string.

☞ Syntax

```
BYTE_LENGTH( string-expression )
```

## Parameters

**string-expression**

The string whose length is to be calculated.

## Returns

INT

## Remarks

Trailing white space characters in the `string-expression` are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the `BYTE_LENGTH` value may differ from the number of characters returned by `CHAR_LENGTH`.

This function supports NCHAR inputs and/or outputs.

**UltraLite:** UltraLite does not support NCHAR inputs and/or outputs.

## Standards

### ANSI/ISO SQL Standard

Not in the standard. The equivalent function is the OCTET\_LENGTH function.

## Example

The following statement returns the value 12:

```
SELECT BYTE_LENGTH( 'Test Message' );
```

## Related Information

[CHAR\\_LENGTH Function \[String\] \[page 363\]](#)

[DATALENGTH Function \[System\] \[page 375\]](#)

[LENGTH Function \[String\] \[page 417\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.10 BYTE\_SUBSTR Function [String]

Returns a substring of a string. The substring is determined using bytes, not characters.

☞ Syntax

```
BYTE_SUBSTR( source-string , start-position [ , length ] )
```

## Parameters

### source-string

The data from which the binary substring is taken.

### start-position

An integer expression indicating the start of the substring. A positive integer starts from the beginning of the data, with the first byte being position 1. A negative integer specifies a substring starting from the end of the data, the final byte being at position -1.

### length

An integer expression indicating the length of the substring. A positive `length` specifies the number of bytes to be taken *starting* at the start position. A negative `length` returns at most `length` bytes up to, and including, the starting position, from the left of the starting position.

## Returns

BINARY or LONG BINARY, depending on the length of the result.

## Remarks

Both `start-position` and `length` can be either positive or negative. Use appropriate combinations of negative and positive numbers, to get a substring from either the beginning or end of the string. If `length` is specified, the maximum length of the substring is `ABS(length)`.

If `start-position` is zero and `length` is non-negative, then a `start-position` value of 1 is used. If `start-position` is zero and `length` is negative, then a start value of -1 is used.

The argument `source-string` can be any data type that can be converted to a binary data type, and is treated as a binary string.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the binary value `0x54657374`, which is the hexadecimal representation of `Test`:

```
SELECT BYTE_SUBSTR( 'Test Message', 1, 4 );
```

## Related Information

[SUBSTRING Function \[String\] \[page 494\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.11 CAST Function [Data Type Conversion]

Returns the value of an expression converted to a supplied data type.

### ☞ Syntax

```
CAST( expression AS datatype )
```

## Parameters

### **expression**

The expression to be converted.

### **datatype**

The data type to cast the expression into. Set the data type explicitly, or specify the %TYPE attribute to set the data type to the data type of a column in a table or view, or to the data type of a variable.

## Returns

Depends on the data type requested.

## Remarks

If you use the CAST function to truncate strings, then the string\_rtruncation database option must be set to OFF; otherwise, there will be an error. Use the LEFT function to truncate strings.

If you do not indicate a length for character string types, then an appropriate length is chosen. If neither precision nor scale is specified for a DECIMAL conversion, then the database server selects appropriate values.

**UltraLite:** It is recommended that you explicitly indicate the precision and scale in your CAST function. The ability to convert depends on the value used in the conversion. The values in the original data type must be compatible with the new data type to avoid generating a conversion error. Use the following chart to determine whether a conversion is supported:

FROM:\nTO:	BIT	TINYINT	UNSIGNED SMALLINT	SMALLINT	UNSIGNED INTEGER	INTEGER	UNSIGNED BIGINT	BIGINT	FLOAT	REAL	DOUBLE	NUMERIC OR DECIMAL	DATE	TIME	DATETIME OR\nTIMESTAMP	TIMESTAMP WITH TIME\nZONE	UNIQUEIDENTIFIER	BINARY OR\nVARBINARY	LONG BINARY	CHAR OR VARCHAR	LONG VARCHAR	ST_GEOMETRY
BIT		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗
TINYINT	⚠		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗
UNSIGNED SMALLINT	⚠	⚠		✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗
SMALLINT	⚠	⚠	⚠		✓	✓	✓	✓	✓	✓	✓	✓	⚠	✗	⚠	✗	✗	✓	✓	✓	✓	✗
UNSIGNED INTEGER	⚠	⚠	⚠	⚠		✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗
INTEGER	⚠	⚠	⚠	⚠	⚠		✓	✓	✓	✓	✓	✓	⚠	✗	⚠	✗	✗	✓	✓	✓	✓	✗
UNSIGNED BIGINT	⚠	⚠	⚠	⚠	⚠	⚠		✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗
BIGINT	⚠	⚠	⚠	⚠	⚠	⚠	⚠		✓	✓	✓	✓	⚠	✗	⚠	✗	✗	✓	✓	✓	✓	✗
FLOAT	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠		⚠	✓	✓	⚠	✗	⚠	✗	✗	✓	✓	✓	✓	✗
REAL	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	✓		✓	✓	⚠	✗	⚠	✗	✗	✓	✓	✓	✓	✗
DOUBLE	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	✓	⚠		✓	⚠	✗	⚠	✗	✗	✓	✓	✓	✓	✗
NUMERIC OR DECIMAL	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	✓	✓	✓		⚠	✗	⚠	✗	✗	✓	✓	✓	✓	✗
DATE	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗		✗	✓	✓	✗	✗	✗	✓	✗	✗
TIME	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗		✓	✓	✗	✗	✓	✓	✗	✗
DATETIME OR\nTIMESTAMP	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓		✓	✗	✗	✓	✓	✓	✗
TIMESTAMP WITH TIME\nZONE	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓		✗	✗	✗	✓	✓	✗
UNIQUEIDENTIFIER	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗		⚠	⚠	✓	✓	✗
BINARY OR\nVARBINARY	✓	✓	✓	✓	✓	✓	✓	✓	⚠	⚠	⚠	✓	✗	✗	✗	✗	✗		✓	✓	✓	✗
LONG BINARY	✓	✓	✓	✓	✓	✓	✓	✓	⚠	⚠	⚠	✓	✗	✗	✗	✗	✗	✓		✓	✓	✗
CHAR OR VARCHAR	✗	✗	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠
LONG VARCHAR	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠
ST_GEOMETRY	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	

Symbol

Compatibility



Always converts



Never converts



Value-dependent

## i Note

In UltraLite:

- To convert between a VARBINARY and a UNIQUEIDENTIFIER, the VARBINARY value must have a 16 byte length.
- To convert between a NUMERIC and a VARBINARY, the NUMERIC source must have a value that can also be cast as a BIGINT.
- To convert from a VARCHAR to an ST\_GEOMETRY, the VARCHAR source must represent a valid geometry in either WKT or EWKT format.
- To convert from a VARBINARY to an ST\_GEOMETRY, the VARBINARY source must represent a valid geometry in WKB format.
- When casting from a WKB or WKT formatted source to an ST\_GEOMETRY, an SRID of 0 is assigned to the ST\_GEOMETRY value. When casting from an ST\_GEOMETRY, VARCHAR values are formatted in EWKT and VARBINARY values are formatted in WKB.

The HEXTOINT and INTTOHEX functions can be used to convert to and from hexadecimal values.

## Standards

### ANSI/ISO SQL Standard

Core Feature. However, in the software, CAST supports a number of data type conversions that are not permitted by the ANSI/ISO SQL Standard. For example, you can CAST an integer value to a DATE type, whereas in the ANSI/ISO SQL Standard this type of conversion is not permitted.

## Example

The following function ensures a string is used as a date:

```
SELECT CAST( '2000-10-31' AS DATE );
```

The value of the expression  $1 + 2$  is calculated, and the result is then cast into a single-character string.

```
SELECT CAST( 1 + 2 AS CHAR );
```

Casting between VARCHAR and ST\_GEOMETRY is usually implicit. For example, the following statement adds values to ST\_GEOMETRY columns using the ST\_POINT function and a VARCHAR. Each value is implicitly cast to an ST\_GEOMETRY data type consistent with the table columns, but results still appear as VARCHAR.

```
INSERT INTO T1 VALUES (2, ST_POINT(1,2,0), 'SRID=2163;Point(1 2)');
```

The following statement casts a value to the data type defined for the BirthDate column (DATE data type) of the Employees table:

```
SELECT CAST ( '1966-10-30' AS Employees.BirthDate%TYPE );
```



**UltraLite:** You can use the CAST function to shorten strings:

```
SELECT CAST ( 'Surname' AS CHAR(5) );
```

## Related Information

[CONVERT Function \[Data Type Conversion\] \[page 367\]](#)

### 1.13.5.2.12 CEILING Function [Numeric]

Returns the first integer that is greater or equal to a given value. For positive numbers, this is known as rounding up.

#### Syntax

```
{ CEILING | CEIL } ( numeric-expression )
```

## Parameters

### **numeric-expression**

The number whose ceiling is to be calculated.

## Returns

DOUBLE

## Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

## Standards

ANSI/ISO SQL Standard

The CEILING function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

## Example

The following statement returns the value 60:

```
SELECT CEILING ( 59.84567 );
```

## Related Information

[FLOOR Function \[Numeric\] \[page 400\]](#)

### 1.13.5.2.13 CHAR Function [String]

Returns the character with the ASCII value of a number.

☞ Syntax

```
CHAR( integer-expression )
```

## Parameters

**integer-expression**

The number to be converted to an ASCII character. The number must be in the range 0 to 255, inclusive.

## Returns

VARCHAR

## Remarks

The character returned corresponds to the supplied numeric expression in the current database character set, according to a binary sort order.

CHAR returns NULL for integer expressions with values greater than 255 or less than zero.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value Y:

```
SELECT CHAR( 89 );
```

## Related Information

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.14 CHAR\_LENGTH Function [String]

Returns the number of characters in a string.

☰ Syntax

```
CHAR_LENGTH ( string-expression )
```

## Parameters

### string-expression

The string whose length is to be calculated.

## Returns

INT

## Remarks

Trailing white space characters are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the value returned by the CHAR\_LENGTH function may differ from the number of bytes returned by the BYTE\_LENGTH function.

You can use the CHAR\_LENGTH function and the LENGTH function interchangeably for CHAR, VARCHAR, LONG VARCHAR, and NCHAR data types. However, you must use the LENGTH function for BINARY and bit array data types. This function supports NCHAR inputs and/or outputs.

**UltraLite:** You can use the CHAR\_LENGTH function and the LENGTH function interchangeably for CHAR, VARCHAR and LONG VARCHAR data types. However, you must use the LENGTH function for BINARY and bit array data types.

## Standards

### ANSI/ISO SQL Standard

CHAR\_LENGTH is a Core Feature. Using CHAR\_LENGTH over an expression of type NCHAR comprises part of optional ANSI/ISO SQL Language Feature F421.

## Example

The following statement returns the value 8:

```
SELECT CHAR_LENGTH( 'Chemical' );
```

## Related Information

[BYTE\\_LENGTH Function \[String\] \[page 355\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.15 CHARINDEX Function [String]

Returns the position of one string in another.

☰ Syntax

```
CHARINDEX( string-expression-1 , string-expression-2 )
```

## Parameters

### **string-expression-1**

The string for which you are searching. The value must be less than 256 bytes.

### **string-expression-2**

The string to be searched.

## Returns

INT

## Remarks

The first character of `string-expression-1` is identified as 1. If the string being searched contains more than one instance of the other string, then the CHARINDEX function returns the position of the first instance.

If the string being searched does not contain the other string, then the CHARINDEX function returns 0.

If any of the arguments are NULL, the result is NULL.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs or outputs.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns last and first names from the Surname and GivenName columns of the Employees table, but only when the last name begins with the letter K:

```
SELECT Surname, GivenName
FROM GROUPO.Employees
WHERE CHARINDEX( 'K', Surname ) = 1;
```

The following results are returned:

Surname	GivenName
Klobucher	James
Kuo	Felicia
Kelly	Moira

## Related Information

[SUBSTRING Function \[String\] \[page 494\]](#)

[REPLACE Function \[String\] \[page 458\]](#)

[LOCATE Function \[String\] \[page 422\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.16 COALESCE Function [Miscellaneous]

Returns the first non-NULL expression from a list. This function is identical to the ISNULL function.

☞ Syntax

```
COALESCE( expression , expression [ , ... ] )
```

## Parameters

**expression**

Any expression.

At least two expressions must be passed into the function, and all expressions must be comparable.

## Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the first non-NULL expression from the list. If the database server cannot find a common comparison type, then an error is returned.

## Remarks

The result is NULL only if all the arguments are NULL.

The parameters can be of any scalar type, but not necessarily same type.

## Standards

### ANSI/ISO SQL Standard

Core Feature.

## Example

The following statement returns the value 34:

```
SELECT COALESCE ( NULL, 34, 13, 0 );
```

## Related Information

[ISNULL Function \[Miscellaneous\] \[page 413\]](#)

## 1.13.5.2.17 CONVERT Function [Data Type Conversion]

Returns an expression converted to a supplied data type.

### Syntax

```
CONVERT( datatype , expression [ , format-style ] )
```

## Parameters

### datatype

The data type to convert the expression into. Set the data type explicitly, or specify the %TYPE attribute to set the data type to the data type of a column in a table or view, or to the data type of a variable.

### expression

The expression to be converted.

**format-style**

The style code to apply to the output value. Use this parameter when converting strings to date or time data types, and vice versa. The table below shows the supported style codes, followed by a representation of the output format produced by that style code. The style codes are separated into two columns, depending on whether the century is included in the output format (for example, 06 versus 2006).

Style code 0 is used if an argument is not provided.

Without century (yy) style codes	With century (yyyy) style codes	Output format
-	0 or 100	Mmm dd yyyy hh:nnAA
1	101	mm/dd/yy[yy]
2	102	[yy]yy.mm.dd
3	103	dd/mm/yy[yy]
4	104	dd.mm.yy[yy]
5	105	dd-mm-yy[yy]
6	106	dd Mmm yy[yy]
7	107	Mmm dd, yy[yy]
8	108	hh:nn:ss
-	9 or 109	Mmm dd yyyy hh:nn:ss:sssAA
10	110	mm-dd-yy[yy]
11	111	[yy]yy/mm/dd
12	112	[yy]yymmdd
-	13 or 113	dd Mmm yyyy hh:nn:ss:sss (24 hour clock, Europe default + milliseconds, 4-digit year )
-	14 or 114	hh:nn:ss:sss (24 hour clock)
-	20 or 120	yyyy-mm-dd hh:nn:ss (24-hour clock, ODBC canonical, 4-digit year)
-	21 or 121	yyyy-mm-dd hh:nn:ss:sss (24 hour clock, ODBC canonical with milliseconds, 4-digit year )

**Returns**

Depends on the data type specified.

**Remarks**

The CONVERT function can be used to convert a string to a DATE, TIME, or TIMESTAMP data type, provided that there is no ambiguity when parsing the string. If *format-style* is specified, then the database server



may use it as a hint on how to parse the string. The database server returns an error if it cannot parse the string unambiguously.

**UltraLite:** This function is similar to the CAST function but allows you to specify a format style to assist with date and time data type conversions.

## Standards

### ANSI/ISO SQL Standard

The CONVERT function is defined in the ANSI/ISO SQL Standard. However, in the Standard the purpose of CONVERT is to perform a transcoding of the input string expression to a different character set, which is implemented in the software as the CSCONVERT function.

## Example

The following statements illustrate the use of format style:

```
SELECT CONVERT ( CHAR( 20 ), OrderDate, 104 ) FROM GROUPO.SalesOrders;
```

### OrderDate

---

16.03.2000

---

20.03.2000

---

23.03.2000

---

25.03.2000

---

...

---

```
SELECT CONVERT ( CHAR( 20 ), OrderDate, 7 ) FROM GROUPO.SalesOrders;
```

### OrderDate

---

Mar 16, 00

---

Mar 20, 00

---

Mar 23, 00

---

Mar 25, 00

---

...

---

The following statement illustrates conversion to an integer and returns the value 5:

```
SELECT CONVERT ( integer, 5.2 );
```

The following statement converts a value to the data type defined for the BirthDate column (DATE data type) of the Employees table:

```
SELECT CONVERT ( Employees.BirthDate%TYPE, '1966-10-30' );
```

## Related Information

[UltraLite date\\_format Creation Option \[page 151\]](#)

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

### 1.13.5.2.18 COS Function [Numeric]

Returns the cosine of the angle in radians given by its argument.

☞ Syntax

```
COS( numeric-expression )
```

## Parameters

**numeric-expression**

The angle, in radians.

## Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

## Standards

**ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the value of the cosine of an angle 0.52 radians:

```
SELECT COS ( 0.52 );
```

## Related Information

[ACOS Function \[Numeric\] \[page 346\]](#)

[COT Function \[Numeric\] \[page 371\]](#)

[SIN Function \[Numeric\] \[page 472\]](#)

[TAN Function \[Numeric\] \[page 500\]](#)

### 1.13.5.2.19 COT Function [Numeric]

Returns the cotangent of the angle in radians given by its argument.

#### ☞ Syntax

```
COT( numeric-expression )
```

## Parameters

### **numeric-expression**

The angle, in radians.

## Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the cotangent value of 0.52:

```
SELECT COT ( 0.52 );
```

## Related Information

[COS Function \[Numeric\] \[page 370\]](#)

[SIN Function \[Numeric\] \[page 472\]](#)

[TAN Function \[Numeric\] \[page 500\]](#)

## 1.13.5.2.20 COUNT Function [Aggregate]

Counts the number of rows in a group depending on the specified parameters.

### Syntax

#### Expressions

```
COUNT( [ * | [ ALL | DISTINCT ] expression ] )
```

#### Window function

```
COUNT( [ * | [ ALL ] expression ] )OVER( window-spec )
```

`window-spec` : see the Remarks section below

#### UltraLite expressions

```
COUNT( [ * | [ DISTINCT ] expression ] )
```

## Parameters

\*

Return the number of rows in each group. COUNT(\*) and COUNT() are semantically equivalent.

#### [ ALL ] expression

Return the number of rows in each group where the value of `expression` is not NULL.

#### DISTINCT expression

Return the number of distinct values of `expression` for all of the rows in each group where `expression` is not NULL.

#### UltraLite expression

Return the number of rows in each group where the value of `expression` is not null.

p

## Returns

The COUNT function returns a value of type INT.

COUNT never returns the value NULL. If a group contains no rows, or if there are no non-NULL values of *expression* in a group, then COUNT returns 0.

## Remarks

In SQL Anywhere, the COUNT function returns a maximum value of 2147483647. Use the COUNT\_BIG function when counting large result sets, the result might have more rows, or there is a possibility of overflow.

Specifying this function with *window-spec* represents usage as a window function in a SELECT statement. As such, elements of *window-spec* can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

## Standards

### ANSI/ISO SQL Standard

Core Feature. When used as a window function, COUNT comprises part of optional ANSI/ISO SQL Language Feature T611, "Basic OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions". The software does not permit the arguments of an aggregate function to include both a column reference from the query block containing the COUNT function, combined with an outer reference.

## Example

The following statement returns each unique city, and the number of employees working in that city:

```
SELECT City, COUNT( * ) FROM GROUPO.Employees GROUP BY City;
```

The following statement returns each unique city, and the number of managers working in that city:

```
SELECT City, COUNT( DISTINCT ManagerID ) FROM GROUPO.Employees GROUP BY City;
```

## Related Information

[AVG Function \[Aggregate\] \[page 353\]](#)

[SUM Function \[Aggregate\] \[page 497\]](#)

[Troubleshooting Database Upgrades: Aggregate Functions and Outer References](#)

### 1.13.5.2.21 COUNT\_UPLOAD\_ROWS function [Aggregate]

Returns a count of the number of rows that will be uploaded in the next synchronization.

☰ Syntax

```
COUNT_UPLOAD_ROWS( pubs, threshold)
```

## Parameters

**pubs**

A comma-separated list of publications to check for rows.

**threshold**

The maximum number of rows to count (a value of 0 corresponds to the maximum limit).

## Returns

INT

## Example

The following returns the total number of rows to upload in `mypub1` and `mypub2`:

```
SELECT COUNT_UPLOAD_ROWS ( 'mypub1, mypub2', 0 );
```

## 1.13.5.22 DATALENGTH Function [System]

Returns the length, in bytes, of the underlying storage for the result of an expression.

☞ Syntax

```
DATALENGTH( expression )
```

### Parameters

**expression**

Usually a column name. If `expression` is a string constant, you must enclose it in quotes.

### Returns

UNSIGNED INT

### Remarks

The return values of the DATALENGTH function are as follows:

Data type	DATALENGTH
BIT	1
TINYINT	1
SMALLINT	2
INTEGER	4
BIGINT	8
REAL	4
DOUBLE	8
TIME	8
DATE	4
TIMESTAMP	8
DATETIME	8
TIMESTAMP WITH TIME ZONE	29
UNIQUEIDENTIFIER	16

Data type	DATALENGTH
CHAR	Length of the data
VARCHAR	Length of the data
BINARY	Length of the data
VARBINARY	Length of the data
NCHAR	Length of the data
NVARCHAR	Length of the data
TEXT	Length of the data
NTEXT	Length of the data
IMAGE	Length of the data
XML	Length of the data

In SQL Anywhere, this function supports NCHAR inputs and outputs.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the length of the longest string in the CompanyName column:

```
SELECT MAX( DATALENGTH( CompanyName ) )
FROM GROUPO.Customers;
```

The following statement returns the length of the string '8sdofinsv8s7a7s7gehe4h':

```
SELECT DATALENGTH( '8sdofinsv8s7a7s7gehe4h' );
```

## 1.13.5.2.23 DATE Function [Date and Time]

Converts the expression into a date, and removes any hours, minutes, or seconds.

☞, Syntax

```
DATE( expression )
```



## Parameters

### **expression**

The value to be converted to date format, typically a string.

## Returns

DATE

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the value 1999-01-02 as a date:

```
SELECT DATE( '1999-01-02 21:20:53' );
```

The following statement returns the create dates of all the objects listed in the SYSOBJECT system view:

```
SELECT DATE( creation_time ) FROM SYS.SYSOBJECT;
```

## Related Information

[UltraLite date\\_order Creation Option \[page 153\]](#)

### 1.13.5.2.24 DATEADD Function [Date and Time]

Returns a TIMESTAMP or TIMESTAMP WITH TIME ZONE value produced by adding a date part to its argument.

#### ⌘ Syntax

```
DATEADD( date-part , integer-expression , timestamp-expression )
```

date-part :

*year* | *quarter* | *month* | *week* | *day* | *dayofyear* | *hour* | *minute* | *second* | *millisecond*  
| *microsecond*

## Parameters

### **date-part**

The date part that *integer-expression* represents.

### **integer-expression**

The number of *date-part* values to be added to *timestamp-expression*. *integer-expression* can be any numeric type, but its value is truncated to an INTEGER. This value can be positive, zero, or negative.

### **timestamp-expression**

The TIMESTAMP or TIMESTAMP WITH TIME ZONE value to be modified.

## Returns

TIMESTAMP WITH TIME ZONE if *timestamp-expression* is a TIMESTAMP WITH TIME ZONE; otherwise  
TIMESTAMP.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the TIMESTAMP value 2016-05-02 00:00:00.000:

```
SELECT DATEADD( month, 12, '2015/05/02' );
```

The following statement returns the TIMESTAMP value 2015-05-02 04:00:00.000:

```
SELECT DATEADD( hour, 4, '2015/05/02' );
```

You can specify a minus sign to subtract from a date or time. For example, to get a timestamp from 31 days ago, you can execute the following:

```
SELECT DATEADD( day, -31, NOW() );
```

The following statement returns the `TIMESTAMP WITH TIME ZONE` value `2015-05-06 11:33:00.000+04:00`:

```
SELECT DATEADD( day, 4, CAST( '2015/05/02 11:33:00.000000+04:00' as TIMESTAMP
WITH TIME ZONE ) );
```

## Related Information

[Specifying Date Parts \[page 328\]](#)

### 1.13.5.2.25 DATEDIFF Function [Date and Time]

Returns the interval between two dates.

#### ☰ Syntax

```
DATEDIFF( date-part , date-expression-1 , date-expression-2 )
```

*date-part* :

*year* | *quarter* | *month* | *week* | *day* | *dayofyear* | *hour* | *minute* | *second* | *millisecond*  
| *microsecond*

## Parameters

### **date-part**

Specifies the date part in which the interval is to be measured.

### **date-expression-1**

The starting date for the interval. This value is subtracted from *date-expression-2* to return the number of *date-parts* between the two arguments.

### **date-expression-2**

The ending date for the interval. *Date-expression-1* is subtracted from this value to return the number of *date-parts* between the two arguments.

## Returns

INT with year, quarter, month, week, day, and dayofyear. BIGINT with hour, minute, second, millisecond, and microsecond.

## Remarks

This function calculates the number of date parts between two specified dates. The result is a signed integer value equal to (date-expression-2 - date-expression-1), in date parts.

The DATEDIFF function results are truncated, not rounded, when the result is not an even multiple of the date part.

When you use **day** as the date part, the DATEDIFF function returns the number of midnights between the two times specified, including the second date but not the first.

When you use **month** as the date part, the DATEDIFF function returns the number of first-of-the-months between two dates, including the second date but not the first.

When you use **week** as the date part, the DATEDIFF function returns the number of Sundays between the two dates, including the second date but not the first.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns 1:

```
SELECT DATEDIFF( hour, '4:00AM', '5:50AM' );
```

The following statement returns 102:

```
SELECT DATEDIFF( month, '1987/05/02', '1995/11/15' );
```

The following statement returns 0:

```
SELECT DATEDIFF( day, '00:00', '23:59' );
```

The following statement returns 4:

```
SELECT DATEDIFF( day,
    '1999/07/19 00:00',
    '1999/07/23 23:59' );
```

The following statement returns 0:

```
SELECT DATEDIFF( month, '1999/07/19', '1999/07/23' );
```

The following statement returns 1:

```
SELECT DATEDIFF( month, '1999/07/19', '1999/08/23' );
```

The following example shows how to use the DATEDIFF function to return the number of milliseconds to do a 3-way join with the GROUPO.Customers table. The output is sent to the database server window:

```
BEGIN
    DECLARE startTime, endTime TIMESTAMP;
    DECLARE rowCount INT;

    SET startTime = CURRENT_TIMESTAMP;
    SELECT count(*) INTO rowCount FROM GROUPO.Customers AS T1, GROUPO.Customers
AS T2, GROUPO.Customers AS T3;
    SET endTime = CURRENT_TIMESTAMP;

    MESSAGE 'Time to count rows: ' || DATEDIFF( MILLISECOND, startTime,
endTime ) || ' ms';
END
```

## Related Information

[Specifying Date Parts \[page 328\]](#)

### 1.13.5.2.26 DATEFORMAT Function [Date and Time]

Returns a string representing a date expression in the specified format.

#### ≡ Syntax

```
DATEFORMAT( datetime-expression , string-expression )
```

## Parameters

### **datetime-expression**

The datetime to be converted.

### **string-expression**

The format of the converted date.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

## Returns

VARCHAR

## Remarks

Any allowable date format can be used for the string-expression.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value Jan 01, 1989:

```
SELECT DATEFORMAT( '1989-01-01', 'Mmm dd, yyyy' );
```

## Related Information

[UltraLite date\\_format Creation Option \[page 151\]](#)

## 1.13.5.2.27 DATENAME Function [Date and Time]

Returns the name of the specified part (such as the month June) of a `TIMESTAMP` or `TIMESTAMP WITH TIME ZONE` value, as a character string.

### Syntax

```
DATENAME( date-part , timestamp-expression )
```

## Parameters

### date-part

The date part to be named.

### timestamp-expression

The `TIMESTAMP` or `TIMESTAMP WITH TIME ZONE` value for which the date part name is to be returned. For meaningful results, `timestamp-expression` should contain the requested `date-part`.

## Returns

VARCHAR

## Remarks

The DATENAME function returns a string, even if the result is numeric, such as 23 for the day.

In SQL Anywhere, English names are returned for an English locale, other names are returned when the locale is not English. For example, use the Language (LANG) connection parameter to specify a different language.

When the date part TZOffset (TZ) is specified, DATENAME returns the offset as a string of the form: { + | - }hh:nn.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

In an English locale, the following statement returns the value `May`:

```
SELECT DATENAME ( month, '1987/05/02' );
```

On SQL Anywhere in a German locale, the value returned is `Mai`. In a Spanish locale, the value returned is `Mayo`. Several locales are supported.

The following statement returns the value `-05:00`:

```
SELECT DATENAME ( TZ, CAST ('2016/02/03 12:02:00-5:00' AS TIMESTAMP WITH TIME ZONE) ) AS TZOffset;
```

## Related Information

[Specifying Date Parts \[page 328\]](#)

[DATEPART Function \[Date and Time\] \[page 384\]](#)

## 1.13.5.2.28 DATEPART Function [Date and Time]

Returns a portion of a `TIMESTAMP` or `TIMESTAMP WITH TIME ZONE` value.

☞ Syntax

```
DATEPART( date-part , timestamp-expression )
```

### Parameters

**date-part**

The date part to be returned.

**timestamp-expression**

The `TIMESTAMP` or `TIMESTAMP WITH TIME ZONE` value for which the part is to be returned.

### Returns

INT

### Remarks

For meaningful results `timestamp-expression` should contain the required `date-part` portion.

The numbers that correspond to week days depend on the setting of the `first_day_of_week` database option. By default, `first_day_of_week` is 7 which means Sunday.

### Standards

**ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following statement returns the value 5:

```
SELECT DATEPART( month , '1987/05/02' );
```



For `TIMESTAMP WITH TIME ZONE` strings, the string value must be cast as follows to ensure conversion to the correct type.

```
SELECT DATEPART( TZOffset, CAST('2015-07-01 12:34:56.789000 +05:30' AS TIMESTAMP WITH TIME ZONE) );
```

The following example creates a table, `TableStatistics`, and inserts into it the total number of sales orders per year as stored in the `SalesOrders` table:

```
CREATE TABLE TableStatistics (
    ID INTEGER NOT NULL DEFAULT AUTOINCREMENT,
    Year INT,
    NumberOrders INT );
INSERT INTO TableStatistics ( Year, NumberOrders )
SELECT DATEPART( Year, OrderDate ), COUNT(*)
FROM GROUPO.SalesOrders
GROUP BY DATEPART( Year, OrderDate );
```

## Related Information

[Specifying Date Parts \[page 328\]](#)

### 1.13.5.2.29 DATETIME Function [Date and Time]

Converts an expression into a `TIMESTAMP` value.

#### ☰ Syntax

```
DATETIME( expression )
```

## Parameters

### **expression**

The expression to be converted. It is generally a string.

## Returns

`TIMESTAMP`

## Remarks

Attempts to convert numerical values return an error.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns a timestamp with value 1998-09-09 12:12:12.000:

```
SELECT DATETIME ( '1998-09-09 12:12:12.000' );
```

## Related Information

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

## 1.13.5.2.30 DAY Function [Date and Time]

Returns the day of the month of its argument as an integer between 1 and 31.

☞ Syntax

```
DAY( date-expression )
```

## Parameters

### date-expression

The date as a DATE data type.

## Returns

SMALLINT

## Remarks

The DAY function returns an integer between 1 and 31, corresponding to the day of the month in the argument.

## Standards

ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 12:

```
SELECT DAY( '2001-09-12' );
```

## 1.13.5.2.31 DAYNAME Function [Date and Time]

Returns the name of the day of the week from a date.

☞ Syntax

```
DAYNAME( date-expression )
```

## Parameters

**date-expression**

The date.

## Returns

VARCHAR

## Remarks

The names are returned as: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday.

SQL Anywhere returns English names for an English locale, and returns other names when the locale is not English. For example, the Language (LANG) connection parameter can be used to specify a different language.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

In an English locale, the following statement returns the value *Saturday*:

```
SELECT DAYNAME ( '1987/05/02' );
```

In a German locale, the value returned is *Samstag*. In an Italian locale, the value returned is *sabato*. Several locales are supported.

## 1.13.5.2.32 DAYS Function [Date and Time]

Manipulates a **TIMESTAMP** or returns the number of days between two **TIMESTAMP** values.

### ☞ Syntax

**Return number of days between 0000-02-29 and a **TIMESTAMP** value**

```
DAYS( timestamp-expression )
```

**Return number of days between two **TIMESTAMP** values**

```
DAYS( timestamp-expression , timestamp-expression )
```

**Add time to a **TIMESTAMP****

```
DAYS( timestamp-expression , integer-expression )
```

## Parameters

### **timestamp-expression**

A `TIMESTAMP` value.

### **integer-expression**

The number of days to be added to the `timestamp-expression`. If the `integer-expression` is negative, the appropriate number of days is subtracted from `timestamp-expression`. If you supply an integer expression, the `timestamp-expression` must be explicitly cast as a `TIME`, `DATE` or `TIMESTAMP`. If `timestamp-expression` is a `TIME` value, the current date is assumed.

## Returns

`TIMESTAMP` when adding time to a timestamp; otherwise, `INTEGER`.

## Remarks

The result of the `DAYS` function depends on its arguments. The `DAYS` function ignores hours, minutes, and seconds in its arguments.

### **Return number of days since 0000-02-29**

If you pass a single `timestamp-expression` to the `DAYS` function, it will return the number of days between 0000-02-29 and `timestamp-expression` as an `INTEGER`.

#### **i Note**

0000-02-29 is not meant to imply an actual date; it is the default date used by the `DAYS` function.

### **Return number of days between two `TIMESTAMP` values**

If you pass two `TIMESTAMP` values to the `DAYS` function, the function returns the integer number of days between them.

You can also use the `DATEDIFF` function to get the interval between two dates.

### **Add time to a `TIMESTAMP`**

If you pass a `TIMESTAMP` value and an integer to the `DAYS` function, the function returns the `TIMESTAMP` result of adding the integer number of days to the `timestamp-expression` argument.

You can also use the `DATEADD` function to add a date part to a `TIMESTAMP`.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the integer 729889:

```
SELECT DAYS( '1998-07-13 06:07:12' );
```

The following statements return the integer value -366, indicating that the second DATE value is 366 days before the first. It is recommended that you use the second example (DATEDIFF):

```
SELECT DAYS( '1998-07-13 06:07:12',  
            '1997-07-12 10:07:12' );
```

```
SELECT DATEDIFF( day,  
               '1998-07-13 06:07:12',  
               '1997-07-12 10:07:12' );
```

The following statements return the TIMESTAMP value 1999-07-14 00:00:00.000. It is recommended that you use the second example (DATEADD):

```
SELECT DAYS( CAST('1998-07-13' AS DATE ), 366 );
```

```
SELECT DATEADD( day, 366, '1998-07-13' );
```

## Related Information

[DATEDIFF Function \[Date and Time\] \[page 379\]](#)

[DATEADD Function \[Date and Time\] \[page 377\]](#)

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

### 1.13.5.2.33 DB\_PROPERTY Function [System]

Returns the value of the specified database property.

#### ⌵ Syntax

```
DB_PROPERTY( { property-id | property-name } [ , database-id | database-  
name ] )
```

#### UltraLite:

```
DB_PROPERTY( property-name )
```

## Parameters

### **property-id**

The database property ID.

### **property-name**

The database property name.

### **database-id**

The database ID number, as returned by the DB\_ID function. Typically, the database name is used.

### **database-name**

The name of the database, as returned by the DB\_NAME function.

## Returns

VARCHAR, LONG VARCHAR

## Remarks

Returns a string.

The current database is used if the second argument is omitted.

**UltraLite:** To set an option in UltraLite, use the SET OPTION statement or your component's API-specific Set Database Option method.

## Privileges

No privileges are required to execute this function for the current database. To execute this function for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

NULL is returned if you specify an invalid parameter value or don't have one of the required system privileges.

**UltraLite:** These privileges do not apply to UltraLite.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the page size of the current database, in bytes:

```
SELECT DB_PROPERTY( 'PageSize' );
```

## Related Information

[UltraLite Database Properties \[page 203\]](#)

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

## 1.13.5.2.34 DEGREES Function [Numeric]

Converts a number from radians to degrees.

☞ Syntax

```
DEGREES( numeric-expression )
```

## Parameters

**numeric-expression**

An angle in radians.

## Returns

DOUBLE

## Remarks

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns the degrees of the angle given by *numeric-expression*. If the parameter is NULL, the result is NULL.



## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 29.79380534680281:

```
SELECT DEGREES ( 0.52 );
```

## 1.13.5.2.35 DIFFERENCE Function [String]

Returns the difference in the SOUNDEX values between the two string expressions.

### Syntax

```
DIFFERENCE ( string-expression-1 , string-expression-2 )
```

## Parameters

### **string-expression-1**

The first SOUNDEX argument.

### **string-expression-2**

The second SOUNDEX argument.

## Returns

SMALLINT

## Remarks

The DIFFERENCE function compares the SOUNDEX values of two strings and evaluates the similarity between them, returning a value from 0 through 4, where 4 is the best match.

This function always returns some value. The result is NULL only if one of the arguments are NULL.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns similarity between the words test and chest:

```
SELECT DIFFERENCE( 'test', 'chest' );
```

## Related Information

[SOUNDEX Function \[String\] \[page 473\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.36 DOW Function [Date and Time]

Returns a number from 1 to 7 representing the day of the week of a date, where Sunday=1, Monday=2, and so on.

☞ Syntax

```
DOW( date-expression )
```

## Parameters

### date-expression

The value (of type DATE) to be evaluated.

## Returns

SMALLINT

## Remarks

The DOW function is not affected by the value specified for the `first_day_of_week` database option. For example, even if `first_day_of_week` is set to Monday, the DOW function returns a 2 for Monday.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 5:

```
SELECT DOW( '1998-07-09' );
```

The following statement returns the value 1:

```
SELECT DOW( CAST( '2010/05/30 11:33:00.000000+04:00' as TIMESTAMP WITH TIME ZONE ) );
```

The following statement queries the Employees table and returns the employee StartDate, expressed as the number of the day of the week:

```
SELECT DOW( StartDate ) FROM GROUPO.Employees;
```

## 1.13.5.2.37 EXP Function [Numeric]

Returns the result of the base of natural logarithms  $e$  raised to the power of the given argument.

### ☞ Syntax

```
EXP( numeric-expression )
```

## Parameters

**numeric-expression**

The exponent.

## Returns

DOUBLE

## Remarks

The EXP function returns the result of raising the base of natural logarithms e by the value specified by *numeric-expression*.

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

## Standards

**ANSI/ISO SQL Standard**

The EXP function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

## Example

The statement returns the value 3269017.3724721107:

```
SELECT EXP( 15 );
```

## 1.13.5.2.38 EXPLANATION Function [Miscellaneous]

Returns the optimization strategy of a SQL statement as a plain text string.

☞ Syntax

```
EXPLANATION( string-expression [ , cursor-type [ , update-status ] ] )
```

## UltraLite:

```
EXPLANATION( string-expression )
```

## Parameters

### string-expression

The SQL statement, which is commonly a SELECT statement, but can also be an UPDATE, MERGE, or DELETE statement.

### cursor-type

A cursor type, expressed as a string. Possible values are asensitive, insensitive, sensitive, or keyset-driven. If `cursor-type` is not specified, asensitive is used by default.

### update-status

A string parameter accepting one of the following values indicating how the optimizer should treat the given cursor:

Value	Description
READ-ONLY	The cursor is read-only.
READ-WRITE (default)	The cursor can be read or written to.
FOR UPDATE	The cursor can be read or written to. This is the same as READ-WRITE.

## Returns

LONG VARCHAR

## Remarks

The execution plan for the query, returned as a string.

The GRAPHICAL\_PLAN function offers significantly greater information about access plans, including system properties that may have affected how the statement was optimized.

This information can help you decide which indexes to add or how to structure your database for better performance.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement passes a SELECT statement as a string parameter and returns the plan for executing the query:

```
SELECT EXPLANATION( 'SELECT * FROM Departments WHERE DepartmentID > 100' );
```

The following statement returns a string containing the short form of the text plan for an INSENSITIVE cursor over the query SELECT \* FROM Departments WHERE ...':

```
SELECT EXPLANATION( 'SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100',  
    'insensitive', 'read-only' );
```

## Related Information

[Execution Plans in UltraLite \[page 575\]](#)

## 1.13.5.2.39 EXTRACT Function [Date and Time]

Returns a date part from a DATE, TIME, TIMESTAMP, or TIMESTAMP WITH TIME ZONE expression.

### Syntax

```
EXTRACT( date-part FROM timestamp-expression )
```

## Parameters

### date-part

The date part to be returned. The valid values are YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE\_HOUR, and TIMEZONE\_MINUTE.

### timestamp-expression

The DATE, TIME, or TIMESTAMP or TIMESTAMP WITH TIME ZONE value.

## Returns

If `date-part` is SECOND, then the function returns a string that includes the fractional second (up to microsecond precision). For all other `date-part` values, the function returns an INTEGER.

## Remarks

The EXTRACT function is similar to the DATEPART function but not completely. The EXTRACT function accepts only a subset of date parts. Also, the two functions return different values when `date-part` is SECOND.

Date parts YY, MM, DD, HH, MI, SS, TZH, and TZM may also be used but do not conform to the SQL Standard.

## Standards

### ANSI/ISO SQL Standard

Core feature.

## Example

The following statement returns 56.789000:

```
SELECT EXTRACT ( SECOND FROM '2015-07-01 12:34:56.789000' );
```

The following statement returns 5:

```
SELECT EXTRACT ( TIMEZONE_HOUR FROM CAST ('2015-07-01 12:34:56.789000 +05:30' AS  
TIMESTAMP WITH TIME ZONE) );
```

The following statement returns 30:

```
SELECT EXTRACT ( TIMEZONE_MINUTE FROM CAST ('2015-07-01 12:34:56.789000 +05:30' AS  
TIMESTAMP WITH TIME ZONE) );
```

The following statement returns 2021:

```
SELECT EXTRACT ( YEAR FROM '21-07-01 12:34:56.345678' );
```

It does so since 21-07-01 represents July 1, 2021.

## 1.13.5.2.40 FLOOR Function [Numeric]

Returns the largest integer not greater than the given number.

☞ Syntax

```
FLOOR( numeric-expression )
```

### Parameters

#### **numeric-expression**

The value to be truncated, typically a fixed numeric type with non-zero scale or an approximate numeric type (DOUBLE, REAL, or FLOAT).

### Returns

DOUBLE

### Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

### Standards

#### **ANSI/ISO SQL Standard**

The FLOOR function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

### Example

The following statement returns a Floor value of 123:

```
SELECT FLOOR (123) ;
```



The following statement returns a Floor value of 123:

```
SELECT FLOOR (123.45);
```

The following statement returns a Floor value of -124:

```
SELECT FLOOR (-123.45);
```

## Related Information

[CEILING Function \[Numeric\] \[page 361\]](#)

### 1.13.5.2.41 GETDATE Function [Date and Time]

Returns the current year, month, day, hour, minute, second, and fraction of a second.

☰ Syntax

```
GETDATE()
```

## Returns

TIMESTAMP

## Remarks

The accuracy is limited by the accuracy of the system clock.

The information the GETDATE function returns is equivalent to the information returned by the NOW function and the CURRENT\_TIMESTAMP special value.

### i Note

If the database is using a simulated time zone, the simulated time zone is used to calculate the results of this function.

## Standards

ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the system date and time:

```
SELECT GETDATE ( );
```

## Related Information

[NOW Function \[Date and Time\] \[page 446\]](#)

### 1.13.5.2.42 GREATER Function [Miscellaneous]

Returns the greater of two parameter values.

≡ Syntax

```
GREATER( expression-1 , expression-2 )
```

## Parameters

### **expression-1**

The first parameter value to be compared.

### **expression-2**

The second parameter value to be compared.

## Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the result in the type used for the comparison. If the database server cannot find a common comparison type, an error is returned.

## Remarks

If the parameters are equal, the first is returned.

Variables defined as type TABLE REF are not supported for this function.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 10:


```
SELECT GREATER ( 10, 5 ) FROM SYS.DUMMY;
```

## Related Information

[LESSER Function \[Miscellaneous\] \[page 418\]](#)

## 1.13.5.2.43 HEXTOINT Function [Data Type Conversion]

Returns the decimal integer equivalent of a hexadecimal string.

 Syntax

```
HEXTOINT( hexadecimal-string )
```

## Parameters

### hexadecimal-string

The string to be converted to an integer.

## Returns

The CAST, CONVERT, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values.

The HEXTOINT function returns as INT the platform-independent SQL INTEGER equivalent of the hexadecimal string. The hexadecimal value represents a negative integer if the 8th digit from the right is one of the digits 8-9 and the uppercase or lowercase letters A-F and the previous leading digits are all uppercase or lowercase letter F. The following is not a valid use of HEXTOINT since the argument represents a positive integer value that cannot be represented as a signed 32-bit integer:

```
SELECT HEXTOINT ( '0x0080000001' );
```

## Remarks

The HEXTOINT function accepts string keycodes or variables consisting only of digits and the uppercase or lowercase letters A-F, with or without a 0x prefix. The following are all valid uses of HEXTOINT:

```
SELECT HEXTOINT ( '0xFFFFFFFF' );
SELECT HEXTOINT ( '0x00000100' );
SELECT HEXTOINT ( '100' );
SELECT HEXTOINT ( '0xffffffff80000001' );
```

The HEXTOINT function removes the 0x prefix, if present. If the data exceeds 8 digits, it must represent a value that can be represented as a signed 32-bit integer value.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 420:

```
SELECT HEXTOINT ( '1A4' );
```

## Related Information

[INTTOHEX Function \[Data Type Conversion\] \[page 410\]](#)

### 1.13.5.2.44 HOUR Function [Date and Time]

Returns the hour component of a `TIMESTAMP` value.

☞ Syntax

```
HOUR( timestamp-expression )
```

## Parameters

**timestamp-expression**

A `TIMESTAMP` value.

## Returns

`SMALLINT`

## Remarks

The value returned is the hour portion of the `TIMESTAMP` expression, a `SMALLINT` value between 0 and 23.

## Standards

**ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the value 21:

```
SELECT HOUR( '1998-07-09 21:12:13' );
```

### 1.13.5.2.45 HOURS Function [Date and Time]

Manipulates a `TIMESTAMP` or returns the number of hours between two `TIMESTAMP` values.

#### ☰ Syntax

**Return number of hours between midnight 0000-02-29 and a `TIMESTAMP` value**

```
HOURS ( timestamp-expression )
```

**Return number of hours between two `TIMESTAMP` values**

```
HOURS ( timestamp-expression , timestamp-expression )
```

**Add hours to a `TIMESTAMP`**

```
HOURS ( time-or-timestamp-expression , integer-expression )
```

## Parameters

### **time-or-timestamp-expression**

A value of type `TIME` or `TIMESTAMP`.

### **timestamp-expression**

A value of type `TIMESTAMP`.

### **integer-expression**

The number of hours to be added to `time-or-timestamp-expression`. If `integer-expression` is negative, the appropriate number of hours is subtracted from `time-or-timestamp-expression`.

## Returns

`INTEGER` when returning the number of hours between two `time-or-timestamp-expression` values.

`TIME` or `TIMESTAMP` when adding time to a `time-or-timestamp-expression`

## Remarks

The result of the HOURS function depends on its arguments.

### Return number of hours since midnight 0000-02-29

If you pass a single `timestamp-expression` to the HOURS function, it will return the number of hours between midnight 0000-02-29 and `timestamp-expression` as an INTEGER.

#### i Note

0000-02-29 is not meant to imply an actual date; it is the default TIMESTAMP value used by the HOURS function.

### Return number of hours between two TIMESTAMP values

If you pass two TIMESTAMP values to the HOURS function, the function returns the integer number of hours between them.

### Add hours to a TIMESTAMP

If you pass a TIMESTAMP value and an INTEGER value to the HOURS function, the function returns the TIMESTAMP result of adding the integer number of hours to `time-or-timestamp-expression` argument. Similarly, if you pass a TIME value as the first argument, a TIME value is returned as the result. This syntax does not support implicit conversion of the first argument. It may be necessary to explicitly cast the first argument to a DATE, TIME or TIMESTAMP value. If the first argument is a DATE, midnight is assumed for the time portion.

You can also use the DATEDIFF and DATEADD functions for these calculations.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statements return the value 4, signifying that the second TIMESTAMP value is four hours after the first. It is recommended that you use the second example (DATEDIFF).

```
SELECT HOURS( '1999-07-13 06:07:12', '1999-07-13 10:07:12' );
SELECT DATEDIFF( hour, '1999-07-13 06:07:12', '1999-07-13 10:07:12' );
```

The following statement returns the value 17517342:

```
SELECT HOURS( '1998-07-13 06:07:12' );
```

The following statements return the datetime 1999-05-13 02:05:07.000. It is recommended that you use the second example (DATEADD).

```
SELECT HOURS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5 );  
SELECT DATEADD( hour, 5, '1999-05-12 21:05:07' );
```

## Related Information

[DATEDIFF Function \[Date and Time\] \[page 379\]](#)

[DATEADD Function \[Date and Time\] \[page 377\]](#)

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

### 1.13.5.2.46 IFNULL Function [Miscellaneous]

Evaluates whether one expression is NULL and returns a value.

☞ Syntax

```
IFNULL( expression-1 , expression-2 [ , expression-3 ] )
```

## Parameters

### expression-1

The expression to be evaluated. Its value determines whether *expression-2* or *expression-3* is returned.

### expression-2

The return value if *expression-1* is NULL.

### expression-3

The return value if *expression-1* is not NULL.

## Returns

The data type returned depends on the data type of *expression-2* and *expression-3*.



## Remarks

If the first expression is the NULL value, then the value of the second expression is returned. If the first expression is not NULL, the value of the third expression is returned. If the first expression is not NULL and there is no third expression, NULL is returned.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value -66:

```
SELECT IFNULL( NULL, -66 );
```

The following statement returns NULL, because the first expression is not NULL and there is no third expression:

```
SELECT IFNULL( -66, -66 );
```

## 1.13.5.2.47 INSERTSTR Function [String]

Inserts a string into another string at a specified position.

☞ Syntax

```
INSERTSTR( integer-expression , string-expression-1 , string-expression-2 )
```

## Parameters

### integer-expression

The position after which the string is to be inserted. Use zero to insert a string at the beginning.

### string-expression-1

The string into which the other string is to be inserted.

### string-expression-2

The string to be inserted.

## Returns

LONG BINARY, LONG VARCHAR, or LONG NVARCHAR, depending on the data type of the input expressions.

## Remarks

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs or outputs.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value `backoffice`:

```
SELECT INSERTSTR( 0, 'office ', 'back' );
```

## Related Information

[STUFF Function \[String\] \[page 493\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.48 INTTOHEX Function [Data Type Conversion]

Returns a string containing the hexadecimal equivalent of an integer.

☞ Syntax

```
INTTOHEX( integer-expression )
```

## Parameters

### **integer-expression**

The integer to be converted to hexadecimal.

## Returns

VARCHAR

## Remarks

The CAST, CONVERT, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the value 0000009c:

```
SELECT INTTOHEX ( 156 );
```

## Related Information

[Converting to and from Hexadecimal Values](#)  
[HEXTOINT Function \[Data Type Conversion\] \[page 403\]](#)

## 1.13.5.2.49 ISDATE Function [Data Type Conversion]

Tests if a string argument can be converted to a date.

☞ Syntax

```
ISDATE( string )
```

### Parameters

**string**

The string to be analyzed to determine if the string represents a valid date.

### Returns

INT

### Remarks

If a conversion is possible, the function returns 1; otherwise, 0 is returned. If the argument is NULL, 0 is returned.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs or outputs.

### Standards

**ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following example imports data from an external file into the sample database, exports rows which contain invalid values, and copies the remaining rows to a permanent table:

```
CREATE GLOBAL TEMPORARY TABLE MyData (
```

```

        person VARCHAR(100),
        birth_date VARCHAR(30),
        height_in_cms VARCHAR(10)
    ) ON COMMIT PRESERVE ROWS;
LOAD TABLE MyData FROM 'exported.dat';
UNLOAD
    SELECT * FROM MyData
    WHERE ISDATE( birth_date ) = 0
OR ISNUMERIC( height_in_cms ) = 0
    TO 'badrows.dat';
INSERT INTO PermData
    SELECT person, birth_date, height_in_cms
    FROM MyData
    WHERE ISDATE( birth_date ) = 1
AND ISNUMERIC( height_in_cms ) = 1;
COMMIT;
DROP TABLE MyData;

```

### 1.13.5.2.50 ISNULL Function [Miscellaneous]

Returns the first non-NULL expression from a list. This function is identical to the COALESCE function.

#### ☰ Syntax

```
ISNULL( expression , expression [ , ... ] )
```

#### Parameters

##### expression

An expression to be tested against NULL.

At least two expressions must be passed into the function, and all expressions must be comparable.

#### Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the first non-NULL expression from the list. If the database server cannot find a common comparison type, then an error is returned.

#### Standards

##### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value -66:

```
SELECT ISNULL( NULL ,-66, 55, 45, NULL, 16 );
```

## Related Information

[COALESCE Function \[Miscellaneous\] \[page 366\]](#)

### 1.13.5.2.51 LCASE Function [String]

Converts all characters in a string to lowercase.

☞ Syntax

```
LCASE( string-expression )
```

## Parameters

**string-expression**

The string to be converted to lowercase.

## Returns

LONG NVARCHAR when used on NCHAR data and LONG VARCHAR when used on CHAR data if the database collation is UCA. Otherwise, the data type is the same as the input data type.

**UltraLite:** The returned data type is the same as the input data type.

## Remarks

The LCASE function is identical to the LOWER function.

## Standards

### ANSI/ISO SQL Standard

Not in the standard. The equivalent function LOWER is a Core Feature.

## Example

The following statement returns the value chocolate:

```
SELECT LCASE( 'ChoCoLatE' );
```

## Related Information

[LOWER Function \[String\] \[page 427\]](#)

[UCASE Function \[String\] \[page 506\]](#)

[UPPER Function \[String\] \[page 508\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.52 LEFT Function [String]

Returns multiple characters from the beginning of a string.

☞ Syntax

```
LEFT( string-expression , integer-expression )
```

## Parameters

### string-expression

The string.

### integer-expression

The number of characters to return.

## Returns

LONG VARCHAR or LONG NVARCHAR

**UltraLite:** LONG VARCHAR

## Remarks

If the string contains multibyte characters, and the proper collation is being used, the number of bytes returned may be greater than the specified number of characters.

You can specify an *integer-expression* that is larger than the value in the argument string expression. In this case, the entire value is returned.

Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the first 5 characters of each Surname value in the Customers table:

```
SELECT LEFT( Surname, 5) FROM GROUPO.Customers;
```

## Related Information

[RIGHT Function \[String\] \[page 461\]](#)

[STRING Function \[String\] \[page 490\]](#)



## 1.13.5.2.53 LENGTH Function [String]

Returns the number of characters in the specified string.

☞ Syntax

```
{ LENGTH | LEN } ( string-expression )
```

### Parameters

**string-expression**

The string.

### Returns

INT

### Remarks

Use this function to determine the length of a string. For example, specify a column name for *string-expression* to determine the length of values in the column.

If the string contains multibyte characters, and the proper collation is being used, LENGTH returns the number of characters, not the number of bytes. If the string is of data type BINARY, the LENGTH function behaves as the BYTE\_LENGTH function.

You can use the LENGTH function and the CHAR\_LENGTH function interchangeably for CHAR, VARCHAR, LONG VARCHAR, and NCHAR data types. However, you must use the LENGTH function for BINARY and bit array data types. This function supports NCHAR inputs and/or outputs.

UltraLite does not support the function name short form *LEN*.

UltraLite does not support NCHAR inputs and/or outputs.

### Standards

#### ANSI/ISO SQL Standard

The LENGTH function is not in the standard; however, its semantics are identical to those of the CHAR\_LENGTH function in the ANSI/ISO SQL Standard. Using LENGTH over a string expression of type NCHAR comprises part of optional ANSI/ISO SQL Language Feature F421.

## Example

The following statement returns the value 9:

```
SELECT LENGTH( 'chocolate' );
```

## Related Information

[International Languages and Character Sets](#)

[BYTE\\_LENGTH Function \[String\] \[page 355\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.54 LESSER Function [Miscellaneous]

Returns the lesser of two parameter values.

☰ Syntax

```
LESSER( expression-1 , expression-2 )
```

## Parameters

### **expression-1**

The first parameter value to be compared.

### **expression-2**

The second parameter value to be compared.

## Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the result in the type used for the comparison. If the database server cannot find a common comparison type, an error is returned.

## Remarks

If the parameters are equal, the first value is returned.

Variables defined as type TABLE REF are not supported for this function.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 5:

```
SELECT LESSER( 10, 5 ) FROM SYS.DUMMY;
```

## Related Information

[GREATER Function \[Miscellaneous\] \[page 402\]](#)

## 1.13.5.2.55 LIST Function [Aggregate]

Returns a delimited list of values for every row in a group.

### ☞ Syntax

```
LIST( [ ALL | DISTINCT ] string-expression [, delimiter-string ] [ ORDER BY  
order-by-expression [ ASC | DESC ] , ... ] )
```

### UltraLite:

```
LIST( [ DISTINCT ] string-expression [ , delimiter-string ] )
```

## Parameters

string-expression

A string expression, usually a column name. When ALL is specified (the default), for each row in the group, the value of `string-expression` is added to the result string, with values separated by `delimiter-string`. When DISTINCT is specified, only unique `string-expression` values are added.

**UltraLite:** For each row in the group, the value of `string-expression` is added to the result string, with values separated by `delimiter-string`.

#### **delimiter-string**

A delimiter string for the list items. The default setting is a comma. There is no delimiter if a value of NULL or an empty string is supplied. The `delimiter-string` must be a constant.

#### **order-by-expression**

Order the items returned by the function. There is no comma preceding this argument, which makes it easy to use in the case where no `delimiter-string` is supplied.

`order-by-expression` cannot be an integer literal. However, it can be a variable that contains an integer literal.

When an ORDER BY clause contains constants, they are interpreted by the optimizer and then replaced by an equivalent ORDER BY clause. For example, the optimizer interprets ORDER BY 'a' as ORDER BY expression.

A query block containing more than one aggregate function with valid ORDER BY clauses can be executed if the ORDER BY clauses can be logically combined into a single ORDER BY clause. For example, the following clauses:

```
ORDER BY expression1, 'a', expression2
```

```
ORDER BY expression1, 'b', expression2, 'c', expression3
```

are subsumed by the clause:

```
ORDER BY expression1, expression2, expression3
```

## Returns

LONG VARCHAR  
LONG NVARCHAR

### **i** Note

UltraLite does not return LONG NVARCHAR.

## Remarks

The LIST function returns the concatenation (with delimiters) of all the non-NULL values of X for each row in the group. If there does not exist at least one row in the group with a definite X-value, then LIST(X) returns the empty string.

NULL values and empty strings are ignored by the LIST function.

A LIST function cannot be used as a window function, but it can be used as input to a window function.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

## Standards

### ANSI/ISO SQL Standard

The software supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions". The software does not permit the arguments of an aggregate function to include both a column reference from the query block containing the LIST function, combined with an outer reference.

## Example

The following statement returns a list of all the street addresses for employees whose given name is Thomas:

```
SELECT LIST( Street ) FROM GROUPO.Employees
WHERE GivenName = 'Thomas';
```

The following statement returns lists of the names of cities delimited by semicolons and their state, organized by state :

```
SELECT LIST( DISTINCT City, ';' ), State FROM GROUPO.Employees
GROUP BY State;
```

The following statement lists employee IDs. Each row in the result set contains a comma-delimited list of employee IDs for a single department.

```
SELECT LIST( EmployeeID )
FROM GROUPO.Employees
GROUP BY DepartmentID;
```

### LIST( EmployeeID )

---

102,105,160,243,247,249,266,278,...

---

129,195,299,467,641,667,690,856,...

---

148,390,586,757,879,1293,1336,...

---

184,207,318,409,591,888,992,1062,...

---

191,703,750,868,921,1013,1570,...

---

The following statement sorts the employee IDs by the last name of the employee:

```
SELECT LIST( EmployeeID ORDER BY Surname ) AS "Sorted IDs"
```

```
FROM GROUPO.Employees
GROUP BY DepartmentID;
```

#### Sorted IDs

```
1013,191,750,921,868,1658,...
1751,591,1062,1191,992,888,318,...
1336,879,586,390,757,148,1483,...
1039,129,1142,195,667,1162,902,...
```

The following statement returns semicolon-separated lists. Note the position of the ORDER BY clause and the list separator:

```
SELECT LIST( EmployeeID, ';' ORDER BY Surname ) AS "Sorted IDs"
FROM GROUPO.Employees
GROUP BY DepartmentID;
```

#### Sorted IDs

```
1013;191;750;921;868;1658;703;...
1751;591;1062;1191;992;888;318;...
1336;879;586;390;757;148;1483;...
1039;129;1142;195;667;1162;902; ...
160;105;1250;247;266;249;445;...
```

Be sure to distinguish the previous statement from the following statement, which returns comma-separated lists of employee IDs sorted by a compound sort-key of ( Surname, ';' ):

```
SELECT LIST( EmployeeID ORDER BY Surname, ';' ) AS "Sorted IDs"
FROM GROUPO.Employees
GROUP BY DepartmentID;
```

**UltraLite:** The following statement returns all street addresses from the Employees table:

```
SELECT LIST( Street ) FROM GROUPO.Employees;
```

## 1.13.5.2.56 LOCATE Function [String]

Returns the position of one string within another.

#### ☰ Syntax

```
LOCATE( string-expression-1, string-expression-2 [, integer-expression ] )
```

## Parameters

### **string-expression-1**

The string to be searched.

### **string-expression-2**

The string to be searched for.

This string is limited to 254 bytes.

### **integer-expression**

The character position in the string to begin the search. The first character is position 1. If the starting offset is negative, the locate function returns the last matching string offset rather than the first. A negative offset indicates how much of the end of the string is to be excluded from the search. The number of bytes excluded is calculated as  $(-1 * \text{offset}) - 1$ .

## Returns

INT

## Remarks

If *integer-expression* is specified, the search starts at that offset into the string.

The first string can be a long string (longer than 255 bytes), but the second is limited to 254 bytes. If a long string is given as the second argument, the function returns a NULL value. If the string is not found, 0 is returned. Searching for a zero-length string will return 1. If any of the arguments are NULL, the result is NULL.

If multibyte characters are used, with the appropriate collation, then the starting position and the return value may be different from the *byte* positions.

If arguments *string-expression-1* and *string-expression-2* are of binary data type, the LOCATE function behaves the same as the BYTE\_LOCATE function.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the value 8:

```
SELECT LOCATE (
    'office party this week - rsvp as soon as possible',
    'party',
    2 );
```

The following statement:

```
BEGIN
    DECLARE STR LONG VARCHAR;
    DECLARE POS INT;
    SET str = 'c:\test\functions\locate.sql';
    SET pos = LOCATE( str, '\', -1 );
    select str, pos,
        SUBSTR( str, 1, pos -1 ) AS path,
        SUBSTR( str, pos +1 ) AS filename;
END;
```

returns the following output:

str	pos	path	filename
c:\test\functions\locate.sql	18	c:\test\functions	locate.sql

## Related Information

[CHARINDEX Function \[String\] \[page 364\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.57 LOG Function [Numeric]

Returns the natural logarithm of a number.

☰ Syntax

```
LOG( numeric-expression )
```

## Parameters

**numeric-expression**

The number.



## Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

## Remarks

The argument is an expression that returns the value of any built-in numeric data type.

## Standards

### ANSI/ISO SQL Standard

The ANSI/ISO SQL Standard defines the natural logarithm function using the keyword LN. The natural logarithm function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

## Example

The following statement returns the natural logarithm of 50:

```
SELECT LOG( 50 );
```

## Related Information

[LOG10 Function \[Numeric\] \[page 425\]](#)

### 1.13.5.2.58 LOG10 Function [Numeric]

Returns the base 10 logarithm of a number.

#### ☰ Syntax

```
LOG10( numeric-expression )
```

## Parameters

### **numeric-expression**

The number.

## Returns

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the parameter is NULL, the result is NULL.

## Remarks

The argument is an expression that returns the value of any built-in numeric data type.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the base 10 logarithm for 50:

```
SELECT LOG10 ( 50 );
```

## Related Information

[LOG Function \[Numeric\] \[page 424\]](#)

## 1.13.5.2.59 LOWER Function [String]

Converts all characters in a string to lowercase.

☰ Syntax

```
LOWER( string-expression )
```

### Parameters

**string-expression**

The string to be converted to lowercase.

### Returns

LONG NVARCHAR when used on NCHAR data

LONG VARCHAR when used on CHAR data if the database collation is UCA

Otherwise, the data type is the same as the input data type

UltraLite returns the same data type as the input data type

### Remarks

The LCASE function is identical to the LOWER function.

### Standards

**ANSI/ISO SQL Standard**

Core Feature. Using LOWER over an expression of type NCHAR comprises part of the optional Language Feature F421.

### Example

The following statement returns the value `chocolate`:

```
SELECT LOWER( 'chOCOLate' );
```

## Related Information

[LCASE Function \[String\] \[page 414\]](#)

[UCASE Function \[String\] \[page 506\]](#)

[UPPER Function \[String\] \[page 508\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.60 LTRIM Function [String]

Removes leading blanks or specified characters from the string.

≡ Syntax

```
LTRIM( string-expression [ , trim-char-set ] )
```

## Parameters

### **string-expression**

The string to be trimmed.

### **trim-char-set**

The set of characters to trim.

## Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

UltraLite returns only VARCHAR or LONG VARCHAR

## Remarks

By default, `trim-char-set` is the space character. You can specify the set of characters to be trimmed.

The actual length of the result is the length of the expression minus the number of characters removed. If all the characters are removed, the result is an empty string.

If the parameter can be null, the result can be null.

If the parameter is null, the result is the null value.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs and `trim-char-set`.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

The TRIM specifications defined by the ANSI/ISO SQL Standard (LEADING and TRAILING) are supplied by the SQL Anywhere LTRIM and RTRIM functions respectively.

## Example

The following statement returns the value `Test Message` with all leading blanks removed:

```
SELECT LTRIM( '      Test Message' );
```

The following statement returns the value `def` after the specified leading characters are removed:

```
SELECT LTRIM('abcabccbade', 'abc' );
```

## Related Information

[RTRIM Function \[String\] \[page 464\]](#)

[TRIM Function \[String\] \[page 504\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.61 MAX Function [Aggregate]

Returns the maximum expression value found in each group of rows.

### Syntax

#### Expression:

```
MAX( [ ALL | DISTINCT ] expression )
```

#### Window function:

```
MAX( [ ALL ] expression ) OVER( window-spec )
```

`window-spec` : see the Remarks section below

#### UltraLite expression

```
MAX( [ DISTINCT ] expression )
```

## Parameters

### expression

The expression for which the maximum value is to be calculated. This is commonly a column name.

### DISTINCT expression

Returns the same as MAX(`expression`), and is included for completeness.

## Returns

The same data type as the argument.

## Remarks

Rows where `expression` is NULL are ignored. Returns NULL for a group containing no rows.

Variables defined as type TABLE REF are not supported for this function.

For simple comparisons of two expressions, you can use the GREATER function.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. This function supports NCHAR inputs and/or outputs.

## Standards

### ANSI/ISO SQL Standard

Core feature. When used as a window function, MAX comprises part of optional ANSI/ISO SQL Language Feature T611, "Basic OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions", nor does it permit the arguments of an aggregate function to include both a column reference from the query block containing the MAX function, combined with an outer reference.

## Example

The following statement returns the value 138948.000, representing the maximum salary in the Employees table:

```
SELECT MAX( Salary )  
FROM GROUP0.Employees;
```


## Related Information

[MIN Function \[Aggregate\] \[page 433\]](#)

[Troubleshooting Database Upgrades: Aggregate Functions and Outer References](#)

## 1.13.5.2.62 MICROSECOND Function [Date and Time]

Returns the microsecond component of a TIMESTAMP expression.

 Syntax

```
MICROSECOND( timestamp-expression )
```

## Parameters

**timestamp-expression**

A TIMESTAMP value.

## Returns

INTEGER

## Remarks

This function returns a value between 0 and 999999 that represents the fraction of a second (also referred to as a microsecond). This function is equivalent to `DATEPART ( MICROSECOND, timestamp-expression )`.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the microsecond value 789012:

```
SELECT MICROSECOND ( '12:34:56.789012' );
```

## 1.13.5.2.63 MILLISECOND Function [Date and Time]

Returns the millisecond component of a `TIMESTAMP` expression.

### ☞ Syntax

```
MILLISECOND( timestamp-expression )
```

## Parameters

### timestamp-expression

A `TIMESTAMP` value.

## Returns

INTEGER



## Remarks

This function returns a value between 0 and 999999 that represents the fraction of a second in milliseconds. If the timestamp contains fractions of a millisecond, then they are rounded down to the millisecond.

This function is equivalent to `DATEPART( MILLISECOND, timestamp-expression )`.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the millisecond value 789:

```
SELECT MILLISECOND( '12:34:56.78901' );
```

## 1.13.5.2.64 MIN Function [Aggregate]

Returns the minimum expression value found in each group of rows.

### Syntax

#### Expression

```
MIN( [ DISTINCT ] expression )
```

#### Window function

```
MIN( [ ALL ] expression )OVER( window-spec )
```

`window-spec` : see the Remarks section below

#### UltraLite expression

```
MIN( [ ALL | DISTINCT ] expression )
```

## Parameters

`expression`

The expression for which the minimum value is to be calculated. This is commonly a column name.

**DISTINCT expression**

Returns the same as MIN( *expression* ), and is included for completeness.

## Returns

The same data type as the argument.

## Remarks

Rows where *expression* is NULL are ignored. Returns NULL for a group containing no rows.

Variables defined as type TABLE REF are not supported for this function.

This function supports NCHAR inputs and/or outputs.

**UltraLite:** UltraLite does not support NCHAR inputs and/or outputs.

Specifying this function with *window-spec* represents usage as a window function in a SELECT statement. As such, elements of *window-spec* can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

## Standards

### ANSI/ISO SQL Standard

Core feature. When used as a window function, MIN comprises part of optional ANSI/ISO SQL Language Feature T611, "Basic OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions", nor does it not permit the arguments of an aggregate function to include both a column reference from the query block containing the MIN function, combined with an outer reference.

## Example

The following statement returns the value 24903.000, representing the minimum salary in the Employees table:

```
SELECT MIN( Salary )
FROM GROUPO.Employees;
```

## Related Information

[MAX Function \[Aggregate\] \[page 429\]](#)

### 1.13.5.2.65 MINUTE Function [Date and Time]

Returns the minute component of a TIMESTAMP value.

≡ Syntax

```
MINUTE( timestamp-expression )
```

## Parameters

**timestamp-expression**

The TIMESTAMP value.

## Returns

SMALLINT

## Remarks

The value returned is the minute portion of the TIMESTAMP expression, a SMALLINT value between 0 and 59.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 22:

```
SELECT MINUTE ( '1998-07-13 12:22:34' );
```

## 1.13.5.2.66 MINUTES Function [Date and Time]

Manipulates a `TIMESTAMP` or returns the number of minute boundaries between two `TIMESTAMP` values.

### Syntax

Return the number of minutes between midnight 0000-02-29 and a `TIMESTAMP` value

```
MINUTES( timestamp-expression )
```

Return the number of minutes between two `TIMESTAMP` values

```
MINUTES( timestamp-expression, timestamp-expression )
```

Add minutes to a `TIMESTAMP` value

```
MINUTES( timestamp-or-time-expression, integer-expression )
```

## Parameters

### `timestamp-expression`

An expression of type `TIMESTAMP`.

### `timestamp-or-time-expression`

An expression of type `TIME` or `TIMESTAMP`.

### `integer-expression`

The number of minutes to be added to `timestamp-or-time-expression`. If `integer-expression` is negative, the appropriate number of minutes is subtracted from `timestamp-or-time-expression`.

## Returns

INTEGER, TIME, or TIMESTAMP, depending on the usage.

## Remarks

The result of the MINUTES function depends on its arguments.

### Return the number of minutes since midnight 0000-02-29

If you pass a single `timestamp-expression` to the MINUTES function, it will return the number of minute boundaries between midnight 0000-02-29 and `timestamp-expression` as an INTEGER.

#### i Note

0000-02-29 is not meant to imply an actual date; it is the default date used by the MINUTES function.

### Return the number of minutes between two TIMESTAMP values

If you pass two TIMESTAMP values to the MINUTES function, the function returns the integer number of minute boundaries between them.

### Add minutes to a TIMESTAMP value

If you pass a TIMESTAMP value and an INTEGER value to the MINUTES function, the function returns the TIMESTAMP result of adding the integer number of minutes to `timestamp-expression` argument. Similarly, if the first argument to MINUTES is a TIME value, then the result is also a TIME value. This syntax does not support implicit conversion of the first argument. It may be necessary to explicitly cast the first argument to a DATE, TIME or TIMESTAMP value. If the first argument is of type DATE, midnight is assumed for the time portion.

Since MINUTES returns an integer, overflow can occur when used with TIMESTAMP values greater than or equal to 4083-03-23 02:08:00.

You can also use the DATEDIFF and DATEADD function for some of the calculations

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns identical values 240, signifying that the second TIMESTAMP value is 240 minutes after the first. It is recommended that you use DATEDIFF.

```
SELECT
```

```
MINUTES ( '1999-07-13 06:07:12',
          '1999-07-13 10:07:12' ),
DATEDIFF ( minute,
          '1999-07-13 06:07:12',
          '1999-07-13 10:07:12' );
```

The following statement returns the value 1051040527:

```
SELECT MINUTES ( '1998-07-13 06:07:12' );
```

The following statements return the `TIMESTAMP` value 1999-05-12 21:10:07.000. The first statement requires an explicit cast of the literal string parameter. It is recommended that you use the second example (`DATEADD`).

```
SELECT MINUTES ( CAST ( '1999-05-12 21:05:07' AS TIMESTAMP ), 5 );
SELECT DATEADD ( minute, 5, '1999-05-12 21:05:07' );
```

## Related Information

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

### 1.13.5.2.67 ML\_GET\_SERVER\_NOTIFICATION function [System]

This function allows UltraLite users to use lightweight polling to query a notifier on a MobiLink server for server-initiated sync requests.

#### Syntax

```
ML_GET_SERVER_NOTIFICATION( notifier , address , key )
```

## Parameters

### notifier

The name of the notifier on the MobiLink server to poll.

### address

The stream parameters, specified as:

```
tcpip{host=pc1;port=1234}
```

### key

Optional. The notification request key.

## Returns

Returns the subject and content of a notification request for the given request key.

## Remarks

If there are no requests for the given request key, or if the notifier name does not exist on the MobiLink server, the result is NULL. If NULL is provided for the request key, then the remote ID of the user is used as the request key. If a request does exist, the resulting message is returned in the form: `[subject]content` (for example, `[sync]profile1`).

This function communicates over the network as it retrieves responses from the MobiLink server. As a result, this function may require a long execution time resulting from network latency. During execution, there may be periods when the function can execute in the background, allowing work to be performed in the runtime on other connections. These periods are not guaranteed however, and depend on the complexity of the SQL. The recommended method for users to retrieve a MobiLink address to use in this function is to use the `sync_profile_option_value` function with an existing synchronization profile to get the value for the Stream profile option. The value returned by this function call can be used directly as the MobiLink address parameter.

## Example

The following statement returns the subject and content of a notification request for the request key MyKey:

```
SELECT ML_GET_SERVER_NOTIFICATION('Notifier1', 'tcpip{host=sap;port=1234}',  
'MyKey');
```

## Related Information

[SYNC\\_PROFILE\\_OPTION\\_VALUE Function \[System\] - UltraLite \[page 499\]](#)

### 1.13.5.2.68 MOD Function [Numeric]

Returns the remainder when one whole number is divided by another.

☰ Syntax

```
MOD( dividend , divisor )
```

## Parameters

### **dividend**

The dividend, or numerator of the division.

### **divisor**

The divisor, or denominator of the division.

## Returns

- SMALLINT
- INT
- NUMERIC

## Remarks

Division involving a negative dividend gives a negative or zero result. The sign of the divisor has no effect.

## Standards

### **ANSI/ISO SQL Standard**

The MOD function is part of optional ANSI/ISO SQL Language Feature T441.

## Example

The following statement returns the value 2:

```
SELECT MOD( 5, 3 );
```

## Related Information

[REMAINDER Function \[Numeric\] \[page 456\]](#)



## 1.13.5.2.69 MONTH Function [Date and Time]

Returns the month of the given date.

☞ Syntax

```
MONTH( date-expression )
```

### Parameters

**date-expression**

A value of type DATE.

### Returns

SMALLINT

### Remarks

The value returned is a number between 1 and 12, corresponding to the month of the given date.

### Standards

**ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following statement returns the value 7:

```
SELECT MONTH( '1998-07-13' );
```

## 1.13.5.2.70 MONTHNAME Function [Date and Time]

Returns the name of the month from a date.

↵ Syntax

```
MONTHNAME( date-expression )
```

### Parameters

**timestamp-expression**

A TIMESTAMP value.

### Returns

VARCHAR

### Remarks

The MONTHNAME function returns a string, even if the result is numeric, such as 2 for the month of February.

In SQL Anywhere, English names are returned for an English locale, other names are returned when the locale is not English. For example, the Language (LANG) connection parameter can be used to specify a different language.

### Standards

**ANSI/ISO SQL Standard**

Not in the standard.

### Example

In an English locale, the following statement returns the value September:

```
SELECT MONTHNAME ( '1998-09-05' );
```

In SQL Anywhere in a French locale, the value returned is `septembre`. In a Spanish locale, the value returned is `septiembre`. Several locales are supported.

## Related Information

[DATEPART Function \[Date and Time\] \[page 384\]](#)

### 1.13.5.2.71 MONTHS Function [Date and Time]

Manipulates a `TIMESTAMP` or returns the number of month boundaries between two `TIMESTAMP` values.

#### ☰ Syntax

Return the number of months between 0000-02 and a `TIMESTAMP` value

```
MONTHS( timestamp-expression )
```

Return the number of months between two `TIMESTAMP` values

```
MONTHS( timestamp-expression, timestamp-expression )
```

Add months to a `TIMESTAMP` value

```
MONTHS( timestamp-expression, integer-expression )
```

## Parameters

### `timestamp-expression`

A date and time of type `TIMESTAMP`.

### `integer-expression`

The integer number of months (of type `SMALLINT`) to be added to the `timestamp-expression`. If `integer-expression` is negative, the appropriate number of months is subtracted from `timestamp-expression`. If you supply an `integer-expression`, the `timestamp-expression` must be explicitly cast as a `TIME`, `DATE` or `TIMESTAMP` data type. If `timestamp-expression` is a `TIME` value, the current month is assumed.

## Returns

`INTEGER` or `TIMESTAMP`, depending on the usage.

## Remarks

The result of the MONTHS function depends on its arguments. The MONTHS function ignores hours, minutes, and seconds in its arguments.

### Return the number of months since 0000-02

If you pass a single `timestamp-expression` to the MONTHS function, it will return the number of month boundaries between 0000-02 and `timestamp-expression` as an INTEGER.

#### i Note

0000-02 is not meant to imply an actual date; it is the default date used by the MONTHS function.

### Return the number of months between two TIMESTAMP values

If you pass two TIMESTAMP values to the MONTHS function, the function returns the integer number of month boundaries between them.

### Add months to a TIMESTAMP value

If you pass a TIMESTAMP value and a SMALLINT value to the MONTHS function, the function returns the TIMESTAMP result of adding the integer number of months to `timestamp-expression`.

You can also use the DATEDIFF and DATEADD functions to perform some of these calculations.

The value of MONTHS is calculated from the number of first days of the month between the two dates.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statements return the value 2, signifying that the second date is two months after the first. It is recommended that you use the second example (DATEDIFF).

```
SELECT MONTHS ( '1999-07-13 06:07:12', '1999-09-13 10:07:12' );
```

```
SELECT DATEDIFF( month,  
    '1999-07-13 06:07:12',  
    '1999-09-13 10:07:12' );
```

The following statement returns the value 23981:

```
SELECT MONTHS ( '1998-07-13 06:07:12' );
```

The following statements return the TIMESTAMP value 1999-10-12 21:05:07.000. It is recommended that you use the second example (DATEADD).

```
SELECT MONTHS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5 );
```

```
SELECT DATEADD( month, 5, '1999-05-12 21:05:07' );
```

## Related Information

[DATEDIFF Function \[Date and Time\] \[page 379\]](#)

[DATEADD Function \[Date and Time\] \[page 377\]](#)

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

## 1.13.5.2.72 NEWID Function [Miscellaneous]

Generates a UUID (Universally Unique Identifier) value.

☰ Syntax

```
NEWID()
```

## Parameters

There are no parameters associated with the NEWID function.

## Returns

UNIQUEIDENTIFIER

## Remarks

The NEWID function can be used in a DEFAULT clause for a column.

The NEWID function is non-deterministic; successive calls will return different values. The query optimizer does not cache the results of the NEWID function.

UUIDs can be used to uniquely identify rows in a table. A value produced on one computer does not match a value produced on another computer, so they can be used as keys in synchronization and replication environments.

UUID values are also referred to as GUID (Globally Unique Identifier) values.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement creates a table named mytab with two columns. Column pk has a unique identifier data type, and assigns the NEWID function as the default value. Column c1 has an integer data type.

```
CREATE TABLE mytab (  
  pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),  
  c1 INT );
```

The following statement returns a unique identifier as a string:

```
SELECT UUIDTOSTR( NEWID() );
```

For example, the value returned might be 96603324-6FF6-49DE-BF7D-F44C1C7E6856.

## Related Information

[The NEWID Default](#)

[STRTOUUID Function \[String\] \[page 491\]](#)

[UUIDTOSTR Function \[String\] \[page 509\]](#)

### 1.13.5.2.73 NOW Function [Date and Time]

Returns the current date and time as a **TIMESTAMP** value. The accuracy is limited by the accuracy of the system clock.

☞ Syntax

```
NOW( [ * ] )
```

## Returns

TIMESTAMP

## Remarks

NOW is equivalent to the GETDATE function and the CURRENT\_TIMESTAMP special value. NOW(\*) and NOW() are equivalent constructions.

Each instance of the NOW function in a request is evaluated at most once. Multiple instances of NOW in the same request may or may not share the identical TIMESTAMP value.

### **i** Note

If the database is using a simulated time zone, the simulated time zone is used to calculate the results of this function.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the current date and time:

```
SELECT NOW ( * );
```

## Related Information

[CURRENT\\_TIMESTAMP Special Value - UltraLite \[page 261\]](#)

[GETDATE Function \[Date and Time\] \[page 401\]](#)

## 1.13.5.2.74 NULLIF Function [Miscellaneous]

Provides an abbreviated CASE expression by comparing expressions.

☞, Syntax

```
NULLIF( expression-1, expression-2 )
```

### Parameters

**expression-1**

An expression to be compared.

**expression-2**

An expression to be compared.

### Returns

Data type of the first argument.

### Remarks

NULLIF compares the values of the two expressions.

If the first expression equals the second expression, NULLIF returns NULL.

If the first expression does not equal the second expression, or if the second expression is NULL, NULLIF returns the first expression.

The NULLIF function provides a short way to write some CASE expressions.

### Standards

**ANSI/ISO SQL Standard**

Core Feature.



## Example

The following statement returns the value a:

```
SELECT NULLIF( 'a', 'b' );
```

The following statement returns NULL:

```
SELECT NULLIF( 'a', 'a' );
```

## Related Information

[CASE Expressions - UltraLite \[page 268\]](#)

### 1.13.5.2.75 PATINDEX Function [String]

Returns an integer representing the starting position of the first occurrence of a pattern in a string.

↵ Syntax

```
PATINDEX( '%pattern%', string-expression )
```

## Parameters

### pattern

The pattern to be searched for. If the leading percent wildcard is omitted, the PATINDEX function returns one (1) if the pattern occurs at the beginning of the string, and zero if it does not.

The pattern uses the same wildcards as the LIKE comparison. These wildcards are listed in the following table.

The pattern for UltraLite uses the wildcards in the following table:

Wildcard	Matches
_ (underscore)	Any one character
% (percent)	Any string of zero or more characters
[]	Any single character in the specified range or set
[^]	Any single character not in the specified range or set

### string-expression

The string to be searched for the pattern.

## Returns

INT

## Remarks

The PATINDEX function returns the starting position of the first occurrence of the pattern. If the pattern is not found, it returns zero (0).

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 2:

```
SELECT PATINDEX( '%hoco%', 'chocolate' );
```

The following statement returns the value 11:

```
SELECT PATINDEX( '%4_5_', '0a1A 2a3A 4a5A' );
```

The following statement returns 14 which is the first non-alphanumeric character in the string expression. The pattern '%[^a-z0-9]%' can be used instead of '%[^a-zA-Z0-9]%' if the database is case insensitive.

```
SELECT PATINDEX( '%[^a-zA-Z0-9]%', 'SQLAnywhere17 has many new features' );
```

The following statement can be used to retrieve everything up to and including the first non-alphanumeric character in a string:

```
SELECT LEFT( @string, PATINDEX( '%[^a-zA-Z0-9]%', @string ) );
```

The following statements create a table, myTable, and populate it with various strings containing alphanumeric characters, spaces (blanks), and non-alphanumeric characters. Then, the SELECT statement and subsequent results show how you can use PATINDEX to find the starting position of spaces and non-alphanumeric characters in the strings:

```
CREATE TABLE myTable( coll LONG VARCHAR );
INSERT INTO myTable (coll) VALUES( 'the quick brown fox jumped over the lazy
dog' ),
( 'the quick brown fox $$$$ jumped over the lazy dog' ),
( 'the quick brown fox 0999 jumped over the lazy dog' ),
( 'the quick brown fox ** jumped over the lazy dog' ),
```

```

( 'thequickbrownfoxjumpedoverthelazydog' ),
( 'thequickbrownfoxjum999pedoverthelazydog' ),
( 'thequick$$$$brownfox' ),
( 'the quick brown fox$$ jumped over the lazy dog' );
SELECT coll,
    //position of first non-alphanumeric character or space:
    PATINDEX( '%[^a-z0-9]%', coll) AS blank_posn,
    //position of first non-alphanumeric char that isn't a space:
    PATINDEX( '%[^ a-z0-9]%', coll) AS non_alpha_char,
    //everything up to and including first non-alphanumeric char that isn't a
space:
    LEFT ( coll, PATINDEX( '%[^ a-zA-Z0-9]%', coll) ) AS left_str,
    //first non-alphanumeric char that isn't a space, and everything to the right:
    SUBSTRING ( coll, PATINDEX( '%[^ a-zA-Z0-9]%', coll) ) AS sub_str
FROM myTable;

```

coll	blank_posn	non_alpha_char	left_str	sub_str
the quick brown fox jumped over the lazy dog	4	0		the quick brown fox jumped over the lazy dog
the quick brown fox \$\$ \$\$ jumped over the lazy dog	4	21	the quick brown fox \$	\$\$\$\$ jumped over the lazy dog
the quick brown fox 0999 jumped over the lazy dog	4	0		the quick brown fox 0999 jumped over the lazy dog
the quick brown fox ** jumped over the lazy dog	4	21	the quick brown fox *	** jumped over the lazy dog
thequickbrownfoxjum- pedoverthelazydog	0	0		thequickbrownfoxjum- pedoverthelazydog
thequickbrownfox- jum999pedoverthela- zydog	0	0		thequickbrownfox- jum999pedoverthela- zydog
thequick\$\$\$\$brown- fox	9	9	thequick\$	\$\$\$\$brownfox
the quick brown fox\$\$ jumped over the lazy dog	4	20	the quick brown fox\$	\$\$ jumped over the lazy dog

## Related Information

[LOCATE Function \[String\] \[page 422\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.76 PI Function [Numeric]

Returns the numeric value PI.

☰, Syntax

```
PI( [ * ] )
```

### Returns

DOUBLE

### Remarks

This function returns a DOUBLE value.

PI(\*) and PI() are semantically equivalent.

### Standards

**ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following statement returns the value 3.141592653(...):

```
SELECT PI ( * );
```

## 1.13.5.2.77 POWER Function [Numeric]

Calculates one number raised to the power of another.

☰, Syntax

```
POWER( numeric-expression-1, numeric-expression-2 )
```

## Parameters

**numeric-expression-1**

The base.

**numeric-expression-2**

The exponent.

## Returns

DOUBLE

## Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If any argument is NULL, the result is a NULL value.

## Standards

**ANSI/ISO SQL Standard**

The POWER function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

## Example

The following statement returns the value 64:

```
SELECT POWER( 2, 6 );
```

## 1.13.5.2.78 QUARTER Function [Date and Time]

Returns a number indicating the quarter of the year from the supplied TIMESTAMP expression.

☞ Syntax

```
QUARTER( timestamp-expression )
```

## Parameters

### **timestamp-expression**

The date you want the quarter for.

## Returns

INTEGER

## Remarks

The quarters are as follows:

Quarter	Period (inclusive)
1	January 1 to March 31
2	April 1 to June 30
3	July 1 to September 30
4	October 1 to December 31

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the value 2:

```
SELECT QUARTER ( '1987/05/02' );
```

## 1.13.5.2.79 RADIANS Function [Numeric]

Converts a number from degrees to radians.

☰ Syntax

```
RADIANS( numeric-expression )
```

### Parameters

**numeric-expression**

A number, in degrees. This angle is converted to radians.

### Returns

DOUBLE

### Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

### Standards

**ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following statement returns a value of approximately 0.5236:

```
SELECT RADIANS ( 30 );
```

## 1.13.5.2.80 REMAINDER Function [Numeric]

Returns the remainder when one whole number is divided by another.

☞ Syntax

```
REMAINDER( dividend, divisor )
```

### Parameters

#### **dividend**

The dividend, or numerator of the division.

#### **divisor**

The divisor, or denominator of the division.

### Returns

- INTEGER
- NUMERIC

### Remarks

You can also use the MOD function to return the remainder.

### Standards

#### **ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following statement returns the value 2:

```
SELECT REMAINDER( 5, 3 );
```



## Related Information

[MOD Function \[Numeric\] \[page 439\]](#)

### 1.13.5.2.81 REPEAT Function [String]

Concatenates a string a specified number of times.

≡ Syntax

```
REPEAT( string-expression, integer-expression )
```

## Parameters

### **string-expression**

The string to be repeated.

### **integer-expression**

The number of times the string is to be repeated. If *integer-expression* is negative, an empty string is returned.

## Returns

LONG VARCHAR

LONG NVARCHAR

UltraLite returns LONG VARCHAR

## Remarks

If the actual length of the result string exceeds the maximum for the return type, an error occurs. The result is truncated to the maximum string size allowed.

The behavior of this function is identical to that of the REPLICATE function.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value `repeatrepeatrepeat`:

```
SELECT REPEAT( 'repeat', 3 );
```

## Related Information

[REPLICATE Function \[String\] \[page 460\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.82 REPLACE Function [String]

Replaces a string with another string, and returns the new results.

### ☰ Syntax

```
REPLACE( original-string, search-string, replace-string )
```

## Parameters

If any argument is NULL, the function returns NULL.

### **original-string**

The string to be searched. This can be any length.

### **search-string**

The string to be searched for and replaced with `replace-string`. This string is limited to 255 bytes. If `search-string` is an empty string, the original string is returned unchanged.

### **replace-string**

The replacement string, which replaces `search-string`. This can be any length. If `replace-string` is an empty string, all occurrences of `search-string` are deleted.

## Returns

LONG BINARY

LONG VARCHAR

LONG NVARCHAR

UltraLite does not return NVARCHAR

## Remarks

This function replaces all occurrences of search-string with replace-string.

If all arguments are of binary data type, the REPLACE function behaves the same as the BYTE\_REPLACE function.

Comparisons are case-sensitive on case-sensitive databases.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value `xx.def.xx.ghi`:

```
SELECT REPLACE ( 'abc.def.abc.ghi', 'abc', 'xx' );
```

The following statement generates a result set containing ALTER PROCEDURE statements which, when executed, would repair stored procedures that reference a table that has been renamed. (To be useful, the table name must be unique.)

```
SELECT REPLACE (
    REPLACE( proc_defn, 'OldTableName', 'NewTableName' ),
    'CREATE PROCEDURE',
    'ALTER PROCEDURE')
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%';
```

## Related Information

[SUBSTRING Function \[String\] \[page 494\]](#)

[CHARINDEX Function \[String\] \[page 364\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.83 REPLICATE Function [String]

Concatenates a string a specified number of times.

☰ Syntax

```
REPLICATE( string-expression, integer-expression )
```

## Parameters

### **string-expression**

The string to be repeated.

### **integer-expression**

The number of times the string is to be repeated.

## Returns

LONG VARCHAR

LONG NVARCHAR

UltraLite does not return NVARCHAR

## Remarks

If the actual length of the result string exceeds the maximum for the return type, an error occurs. The result is truncated to the maximum string size allowed.

The behavior of this function is identical to that of the REPEAT function.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value `repeatrepeatrepeat`:

```
SELECT REPLICATE( 'repeat', 3 );
```

## Related Information

[REPEAT Function \[String\] \[page 457\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.84 RIGHT Function [String]

Returns the rightmost characters of a string.

☞ Syntax

```
RIGHT( string-expression, integer-expression )
```

## Parameters

### **string-expression**

The string to return the rightmost characters for.

### **integer-expression**

The number of characters at the end of the string to return.

## Returns

LONG VARCHAR

LONG NVARCHAR

UltraLite does not return NVARCHAR

## Remarks

If the string contains multibyte characters, the number of bytes returned may be greater than the specified number of characters.

You can specify an *integer-expression* that is larger than the value in the column. In this case, the entire value is returned.

This function supports NCHAR inputs and/or outputs. Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the last 5 characters of each Surname value in the Customers table:

```
SELECT RIGHT( Surname, 5 ) FROM GROUPO.Customers;
```

## Related Information

[International Languages and Character Sets](#)

[LEFT Function \[String\] \[page 415\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.85 ROUND Function [Numeric]

Rounds the `numeric-expression` to the specified `integer-expression` amount of places after the decimal point.

### ☞ Syntax

```
ROUND( numeric-expression, integer-expression )
```

## Parameters

### **numeric-expression**

The number, passed into the function, to be rounded.

### **integer-expression**

A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative expression specifies the number of significant digits to the left of the decimal point at which to round.

## Returns

NUMERIC

## Remarks

The result of this function is either numeric or double. When there is a numeric result and the integer `integer-expression` is a negative value, the precision is increased by one.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the value 123.200:

```
SELECT ROUND( 123.234, 1 );
```

## Related Information

[TRUNCNUM Function \[Numeric\] \[page 505\]](#)

### 1.13.5.2.86 RTRIM Function [String]

Removes trailing blanks or specified characters from the string.

☞ Syntax

```
RTRIM( string-expression [ , trim-char-set ] )
```

## Parameters

### **string-expression**

The string to be trimmed.

### **trim-char-set**

The set of characters to trim.

## Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

UltraLite VARCHAR and LONG VARCHAR



## Remarks

By default, `trim-char-set` is the space character. You can specify the set of characters to be trimmed.

The actual length of the result is the length of the expression minus the number of characters removed. If all the characters are removed, the result is an empty string.

If the argument is null, the result is the NULL value.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs and `trim-char-set`.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

The TRIM specifications defined by the ANSI/ISO SQL Standard (LEADING and TRAILING) are supplied by the LTRIM and RTRIM functions, respectively, that are provided in the software.

## Example

The following statement returns the string `Test Message`, with all trailing blanks removed:

```
SELECT RTRIM( 'Test Message   ' );
```

The following statement returns the value `def` after the specified trailing characters are removed:

```
SELECT RTRIM('defabcabccba', 'abc' );
```

## Related Information

[TRIM Function \[String\] \[page 504\]](#)

[LTRIM Function \[String\] \[page 428\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.87 SECOND Function [Date and Time]

Returns the seconds value of the `TIMESTAMP` argument.

≡ Syntax

```
SECOND( timestamp-expression )
```

### Parameters

**timestamp-expression**

The `TIMESTAMP` value.

### Returns

`SMALLINT`

### Remarks

Returns a number from 0 to 59 corresponding to the seconds component of the given `TIMESTAMP` argument value.

### Standards

**ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following statement returns the value 25:

```
SELECT SECOND( '1998-07-13 21:21:25' );
```

## 1.13.5.2.88 SECONDS Function [Date and Time]

Manipulates a `TIMESTAMP` or returns the number of second boundaries between two `TIMESTAMP` values.

### ☰ Syntax

Return the number of seconds between midnight 0000-02-29 and a `TIMESTAMP` value

```
SECONDS( timestamp-expression )
```

Return the number of seconds between two `TIMESTAMP` values

```
SECONDS( timestamp-expression, timestamp-expression )
```

Add seconds to a `TIMESTAMP` value

```
SECONDS( time-or-timestamp-expression, integer-expression )
```

## Parameters

### `timestamp-expression`

A `TIMESTAMP` value.

### `time-or-timestamp-expression`

A value of type `TIME` or `TIMESTAMP`.

### `integer-expression`

The number of seconds to be added to the `time-or-timestamp-expression`. If `integer-expression` is negative, the appropriate number of seconds is subtracted from `time-or-timestamp-expression`. If you supply an integer expression, the `time-or-timestamp-expression` must be explicitly cast as a `TIME`, `DATE`, or `TIMESTAMP` data type. If `time-or-timestamp-expression` is a `DATE` type, its time portion is assumed to be midnight.

## Returns

`UNSIGNED BIGINT` when returning the number of seconds between midnight 0000-02-29 and a `TIMESTAMP` value

`SIGNED BIGINT` when returning the number of seconds between two `TIMESTAMP` values.

`TIME` or `TIMESTAMP` when adding seconds to a `TIMESTAMP` value.

## Remarks

The result of the `SECONDS` function depends on its arguments.

### Return the number of seconds between midnight 0000-02-29 and a TIMESTAMP value

If you pass a single `timestamp-expression` to the SECONDS function, it will return the number of second boundaries between midnight 0000-02-29 and `timestamp-expression` as an UNSIGNED BIGINT.

#### **i** Note

0000-02 is not meant to imply an actual date; it is the default date used by the SECONDS function.

### Return the number of seconds between two TIMESTAMP values

If you pass two TIMESTAMP values to the SECONDS function, the function returns the integer number of second boundaries between them as a SIGNED BIGINT value.

### Add seconds to a TIMESTAMP value

If you pass a TIMESTAMP value and a INTEGER value to the SECONDS function, the function returns the TIMESTAMP result of adding the integer number of seconds to `time-or-timestamp-expression`. Similarly, if you pass a TIME value to the SECONDS function, the function returns a value of type TIME.

You can also use the DATEDIFF and DATEADD functions to perform some of these calculations.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns identical values 14400, signifying that the second TIMESTAMP value is 14400 seconds after the first:

```
SELECT
  SECONDS ( '1999-07-13 06:07:12',
            '1999-07-13 10:07:12' ),
  DATEDIFF( second,
            '1999-07-13 06:07:12',
            '1999-07-13 10:07:12' );
```

The following statement returns the value 63062431632:

```
SELECT SECONDS ( '1998-07-13 06:07:12' );
```

The following statements return the TIMESTAMP value 1999-05-12 21:05:12.000:

```
SELECT SECONDS ( CAST( '1999-05-12 21:05:07' AS TIMESTAMP ), 5 );
SELECT DATEADD ( second, 5, '1999-05-12 21:05:07' );
```

## Related Information

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

[DATEADD Function \[Date and Time\] \[page 377\]](#)

[DATEDIFF Function \[Date and Time\] \[page 379\]](#)

### 1.13.5.2.89 SHORT\_PLAN function [Miscellaneous]

Returns a short description of the UltraLite plan optimization strategy of a SQL statement, as a string. The description is the same as that returned by the EXPLANATION function.

☞ Syntax

```
SHORT_PLAN( string-expression )
```

## Parameters

**string-expression**

The SQL statement, which is commonly a SELECT statement, but can also be an UPDATE or DELETE statement.

## Returns

LONG VARCHAR

## Remarks

For some queries, the execution plan for UltraLite may differ from the plan selected for SQL Anywhere.

## Example

The following statement passes a SELECT statement as a string parameter and returns the plan for executing the query:

```
SELECT SHORT_PLAN (
    'SELECT * FROM WHERE DepartmentID > 100' );
```

## Related Information

[EXPLANATION Function \[Miscellaneous\] \[page 396\]](#)

### 1.13.5.2.90 SIGN Function [Numeric]

Returns the sign (positive or negative) of the given number.

≡ Syntax

```
SIGN( numeric-expression )
```

## Parameters

**numeric-expression**

The number for which the sign is to be returned. *numeric-expression* may be of type INTEGER, DOUBLE, or NUMERIC.

## Returns

SMALLINT

## Remarks

For negative numbers, the SIGN function returns -1.

For zero, the SIGN function returns 0.

For positive numbers, the SIGN function returns 1.

## Standards

**ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the value -1:

```
SELECT SIGN( -550 );
```

## 1.13.5.2.91 SIMILAR Function [String]

Returns a number indicating the similarity between two strings.

☞ Syntax

```
SIMILAR( string-expression-1, string-expression-2 )
```

## Parameters

### **string-expression-1**

The first string to be compared.

### **string-expression-2**

The second string to be compared.

## Returns

SMALLINT

## Remarks

The function returns an integer between 0 and 100 representing the similarity between the two strings. The result can be interpreted as the percentage of characters matched between the two strings. A value of 100 indicates that the two strings are identical.

This function can be used to correct a list of names (such as customers). Some customers may have been added to the list more than once with slightly different names. You can use the SIMILAR function to find similar customer names by joining the customer table to itself, producing a report of all similarities greater than 90 percent, but less than 100 percent.

The calculation performed for the SIMILAR function is more complex than just the number of characters that match.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 75, indicating that the two values are 75% similar:

```
SELECT SIMILAR( 'toast', 'coast' );
```

## Related Information

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.92 SIN Function [Numeric]

Returns the sine of a number.

☰ Syntax

```
SIN( numeric-expression )
```

## Parameters

### numeric-expression

The angle, in radians.

## Returns

DOUBLE



## Remarks

The SIN function returns the sine of the argument, where the argument is an angle expressed in radians. The SIN and ASIN functions are inverse operations.

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the SIN value of 0.52:

```
SELECT SIN( 0.52 );
```

## Related Information

[ASIN Function \[Numeric\] \[page 349\]](#)

[COS Function \[Numeric\] \[page 370\]](#)

[COT Function \[Numeric\] \[page 371\]](#)

[TAN Function \[Numeric\] \[page 500\]](#)

## 1.13.5.2.93 SOUNDEX Function [String]

Returns a number representing the sound of a string.

☰ Syntax

```
SOUNDEX( string-expression )
```

## Parameters

### **string-expression**

The string to be evaluated.

## Returns

SMALLINT

## Remarks

The SOUNDEX function value for a string is based on the first letter and the next three consonants other than H, Y, and W. Vowels in *string-expression* are ignored unless they are the first letter of the string. Doubled letters are counted as one letter. For example, the word "apples" is based on the letters A, P, L, and S.

Multibyte characters are ignored by the SOUNDEX function.

Although it is not perfect, the SOUNDEX function normally returns the same number for words that sound similar and that start with the same letter.

The SOUNDEX function works best with English words. It is less useful for other languages.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns two identical numbers, 3827, representing the sound of each name:

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' );
```

## Related Information

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.94 SPACE Function [String]

Returns a specified number of spaces.

☞ Syntax

```
SPACE( integer-expression )
```

### Parameters

**integer-expression**

The number of spaces to return.

### Returns

LONG VARCHAR

### Remarks

If *integer-expression* is negative, a null string is returned.

### Standards

**ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following statement returns a string containing 10 spaces:

```
SELECT SPACE ( 10 );
```

## Related Information

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.95 SQRT Function [Numeric]

Returns the square root of a number.

≡ Syntax

```
SQRT( numeric-expression )
```

## Parameters

**numeric-expression**

The number for which the square root is to be calculated.

## Returns

DOUBLE

## Remarks

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

## Standards

### ANSI/ISO SQL Standard

The SQRT function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

## Example

The following statement returns the value 3:

```
SELECT SQRT ( 9 );
```

## 1.13.5.2.96 ST\_AsBinary Function [Spatial] - UltraLite

Returns a binary string representing the specified geometry.

### ⌘ Syntax

```
ST_AsBinary (geometry-expression)
```

## Returns

**BINARY**

Returns the WKB representation of the *geometry-expression*.

## Remarks

The output format is WKB as defined by OGC SFS 1.1.

This format does not contain Z and M values.

## Example

The following statement returns the result 0x010100000000000000000000f03f0000000000000040:

```
SELECT ST_AsBinary(ST_Point(1.0, 2.0, 4326))
```

The server implicitly invokes the ST\_AsBinary function when converting geometries to BINARY. For example, the following statement returns the result 0x010100000000000000000000f03f0000000000000040:

```
SELECT CAST(ST_Point(1.0, 2.0, 4326) AS BINARY(50))
```

## 1.13.5.2.97 ST\_AsExtText Function [Spatial] - UltraLite

Returns a binary string representing the specified geometry.

☞, Syntax

```
ST_AsExtText(geometry-expression)
```

### Returns

VARCHAR

Returns the EWKT representation of the *geometry-expression*.

### Remarks

The output format is EWKT.

### Example

The following statement returns the result SRID=4326;Point(1 2), with the SRID included as a prefix:

```
SELECT ST_AsExtText(ST_Point(1.0, 2.0, 4326))
```

The ST\_AsExtText() function is implicitly invoked when converting geometries to VARCHAR types. For example, the following statement returns the result SRID=4326;Point(1 2):

```
SELECT CAST(ST_Point(1.0, 2.0, 4326) AS VARCHAR(25))
```

## 1.13.5.2.98 ST\_AsText Function [Spatial] - UltraLite

Returns a binary string representing the specified geometry.

☞, Syntax

```
ST_AsText(geometry-expression)
```

## Returns

VARCHAR

Returns the WKT representation of the *geometry-expression*.

## Remarks

The output format is WKT as defined by OGC SFS 1.1.

## Example

The following statement returns the result Point (1 2):

```
SELECT ST_AsText(ST_Point(1.0, 2.0, 4326))
```

## 1.13.5.2.99 ST\_Distance Function [Spatial] - UltraLite

Returns the smallest distance between two specified geometry values.

☰ Syntax

```
ST_Distance( geo1, geo2 )
```

## Parameters

Name	Type	Description
<i>geo1</i>	ST_Geometry	The first geometry value to be used to calculate the distance between two geometry values.
<i>geo2</i>	ST_Geometry	The second geometry value to be used to calculate the distance between two geometry values.

## Returns

DOUBLE

Returns the smallest distance between the specified geometry values.

## Remarks

If the points are in SRID 4326, the units are in meters.

## Example

The following statement returns the result 3367142.4632130372:

```
SELECT ST_Distance(ST_Point(-79.38,43.65,4326),ST_Point(-123.1,49.28,4326))
```

## 1.13.5.2.100 ST\_Equals Function [Spatial] - UltraLite

Tests whether an ST\_Geometry value is spatially equal to another ST\_Geometry value. Two geometry values can be considered equal if they have the same x and y coordinates and are in the same reference system.

☞ Syntax

```
ST_Equals( geo1, geo2 )
```

## Parameters

Name	Type	Description
geo1	ST_Geometry	The first geometry value to be compared.
geo2	ST_Geometry	The second geometry value to be compared.



## Returns

### BIT

Returns 1 if the two geometry values are spatially equal, otherwise 0.

## Remarks

The test may be limited by the resolution of the spatial reference system or the accuracy of the data.

The ST\_Equals function defines the semantics used for comparison predicates (= and <>), IN list predicates, DISTINCT, and GROUP BY.

## Example

The following statement returns the result 1:

```
SELECT ST_Equals(ST_Point(1,1,4326),ST_Point(1,1,4326))
```

## 1.13.5.2.101 ST\_IntersectsRect Function [Spatial] - UltraLite

Tests if a point is located within the box defined by the two points specified as min and max.

### ☰ Syntax

```
ST_IntersectsRect(location,min,max)
```

## Parameters

Name	Type	Description
<code>location</code>	ST_Geometry	The point to be tested.
<code>min</code>	ST_Geometry	The minimum point value used to define the box.
<code>max</code>	ST_Geometry	The maximum point value used to define the box.

## Returns

### BIT

Returns 1 if `location` intersects with the specified box, otherwise 0.

## Remarks

None.

## Example

The following statement returns the result 1:

```
SELECT ST_IntersectsRect(ST_Point(1,1,4326),ST_Point(0,0,4326),
ST_Point(3,3,4326))
```

## 1.13.5.2.102 ST\_Point Function [Spatial] - UltraLite

Constructs a point based on x and y coordinates.

### Syntax

```
ST_Point(x, y, SRID)
```

## Parameters

Name	Type	Description
<code>x</code>	DOUBLE	The x coordinate to use to construct the point.
<code>y</code>	DOUBLE	The y coordinate to use to construct the point.
<code>SRID</code>	INTEGER	The SRID value associated with the point.

## Returns

**ST\_Point**

Returns an ST\_Geometry value created from the input string.

## Remarks

None.

## Example

The following statement creates a point at (10.0,20.0) in the 2163 reference system:

```
SELECT ST_Point(10.0,20.0,2163)
```

## 1.13.5.2.103 ST\_PointFromExtText Function [Spatial] - UltraLite

Returns an ST\_Geometry value, which is transformed from a VARCHAR value containing the EWKT representation of an ST\_Geometry.

☰, Syntax

```
ST_PointFromExtText(ewkt)
```

## Parameters

Name	Type	Description
<code>ewkt</code>	VARCHAR	The EWKT representation.

## Returns

**ST\_Geometry**

Returns an ST\_Geometry value created from the input string.

## Remarks

None.

## Example

The following statement returns the result SRID=4326;Point( 10 20 ) to show that a point has been created at (10,20) in the 4326 reference system:

```
SELECT ST_PointFromExtText('SRID=4326;Point(10 20)')
```

## 1.13.5.2.104 ST\_PointFromText Function [Spatial] - UltraLite

Returns an ST\_Geometry value, which is transformed from a VARCHAR value containing the WKT representation of an ST\_Geometry.

☰ Syntax

```
ST_PointFromText(wkt, srid)
```

## Parameters

Name	Type	Description
<code>wkt</code>	VARCHAR	The WKT representation.
<code>srid</code>	INT	The spatial reference system identifier of the result is indicated by the SRID parameter.

## Returns

### ST\_Geometry

Returns an ST\_Geometry value created from the input string.

The spatial reference system identifier of the result is the given by parameter `srid`.

## Remarks

None.

## Example

The following statement returns the result SRID=4326;Point( 10 20 ) to show that a point has been created at (10, 20) in the 4326 reference system:

```
SELECT ST_PointFromText('Point(10 20)',4326)
```

## 1.13.5.2.105 ST\_PointFromWKB Function [Spatial] - UltraLite

Returns an ST\_Geometry value, which is transformed from a BINARY value containing the WKB representation of an ST\_Geometry.

☰ Syntax

```
ST_PointFromWKB(wkb, srid)
```

## Parameters

Name	Type	Description
wkb	BINARY	The WKB representation.
srid	INTEGER	The SRID value associated with the point.

## Returns

ST\_Geometry

Returns an ST\_Geometry value created from the input string.

## Remarks

None.

## Example

The following statement returns the result (1.0, 2.0, 4326):

```
SELECT ST_PointFromWKB(0x010100000000000000000000f03f0000000000000040,4326)
```

## 1.13.5.2.106 ST\_SRID Function [Spatial] - UltraLite

Retrieves the spatial reference system (SRID) associated with the geometry value.

☰, Syntax

```
ST_SRID( geo1, srid )
```

## Parameters

Name	Type	Description
<code>geo1</code>	ST_Geometry	The point value.
<code>srid</code>	INTEGER	The SRID value associated with the point.

## Returns

INT

Returns the SRID of the geometry.

## Remarks

None.

## Example

The following statement returns the result 4326:

```
SELECT ST_SRID( ST_Point ( 10, 20, 4326 ) );
```

## 1.13.5.2.107 ST\_X Function [Spatial] - UltraLite

Returns the x coordinate of the ST\_Geometry value.

☞ Syntax

```
ST_X(geo1)
```

### Parameters

Name	Type	Description
geo1	ST_Geometry	The ST_Geometry value from which to determine the x coordinate.

### Returns

**DOUBLE**

Returns the x coordinate of the ST\_Geometry value.

### Remarks

None.

### Example

The following statement returns the result 10.0:

```
SELECT ST_X(ST_Point(10.0,20.0,4326))
```

## 1.13.5.2.108 ST\_Y Function [Spatial] - UltraLite

Returns the y coordinate of the ST\_Geometry value.

☰ Syntax

```
ST_Y(geo1)
```

### Parameters

Name	Type	Description
geo1	ST_Geometry	The ST_Geometry value from which to determine the y coordinate.

### Returns

**DOUBLE**

Returns the y coordinate of the ST\_Geometry value.

### Remarks

None.

### Example

The following example returns the result 20.0:

```
SELECT ST_Y(ST_Point(10.0,20.0,4326))
```



## 1.13.5.2.109 STR Function [String]

Returns the string equivalent of a number.

### ☞ Syntax

```
STR( numeric-expression [, length [, decimal ] ] )
```

## Parameters

### **numeric-expression**

Any approximate numeric (float, real, or double precision) expression between -1E126 and 1E127.

### **length**

The number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, and blanks). The default is 10.

### **decimal**

The number of decimal digits to be returned. The default is 0.

## Returns

VARCHAR

## Remarks

If the integer portion of the number cannot fit in the length specified, then the result is a string of the specified length containing all asterisks. For example, the following statement returns \*\*\*.

```
SELECT STR( 1234.56, 3 );
```

### **i Note**

The maximum length that is supported is 128. Any length that is not between 1 and 128 yields a result of NULL.

## Standards

ANSI/ISO SQL Standard

A feature of the standard.

## Example

The following statement returns a string of six spaces followed by 1235, for a total of ten characters:

```
SELECT STR( 1234.56 );
```

The following statement returns the result 1234.6:

```
SELECT STR( 1234.56, 6, 1 );
```

## Related Information

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.110 STRING Function [String]

Concatenates one or more strings into one large string.

☞ Syntax

```
STRING( string-expression [, ... ] )
```

## Parameters

### **string-expression**

The string to be evaluated.

If only one argument is supplied, it is converted into a single expression. If multiple arguments are supplied, they are concatenated into a single string.

## Returns

LONG VARCHAR, LONG NVARCHAR, or LONG BINARY, depending on the data type of the input expression.

## Remarks

Numeric or date parameters are converted to strings before concatenation. The STRING function can also be used to convert any single expression to a string by supplying that expression as the only parameter.

If all parameters are NULL, STRING returns NULL. If any parameters are non-NULL, then any NULL parameters are treated as empty strings.

UltraLite does not support NCHAR inputs and/or outputs.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value `testing123`:

```
SELECT STRING( 'testing', NULL, 123 );
```

## Related Information

[STR Function \[String\] \[page 489\]](#)

### 1.13.5.2.111 STRTOUUID Function [String]

Converts a string value to a unique identifier (UUID or GUID) value.

≡ Syntax

```
STRTOUUID( string-expression )
```

## Parameters

`string-expression`

A string in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

## Returns

UNIQUEIDENTIFIER

## Remarks

Converts a string in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`, where `x` is a hexadecimal digit, to a unique identifier value.

This function is useful for inserting UUID values into a database.

If the string is not a valid UUID string, a conversion error is returned unless the `conversion_error` option is set to OFF, in which case it returns NULL. This function supports NCHAR inputs and/or outputs. Curly braces can be used as the first and last characters in the `string-expression`. In databases created before version 9.0.2, the UNIQUEIDENTIFIER data type was defined as a user-defined data type and the STRTOUUID and UUIDTOSTR functions were needed to convert between binary and string representations of UUID values. In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type was changed to a native data type and the database server carries out conversions as needed. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions.

**UltraLite:** In databases created before version 9.0.2, the STRTOUUID and UUIDTOSTR functions were needed to convert between binary and string representations of UUID values. In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type was changed to a native data type. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statements are equivalent and return the result `0x6c2b64a93c6f47dc901536b9ed49fec2`:

```
SELECT STRTOUUID ( '6c2b64a9-3c6f-47dc-9015-36b9ed49fec2' );
SELECT STRTOUUID ( '{6c2b64a9-3c6f-47dc-9015-36b9ed49fec2}' );
```

## Related Information

[SQL Data Types \[page 288\]](#)

[UUIDTOSTR Function \[String\] \[page 509\]](#)

[NEWID Function \[Miscellaneous\] \[page 445\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.112 STUFF Function [String]

Deletes multiple characters from one string and replaces them with another string.

☰ Syntax

```
STUFF( string-expression-1, start, length, string-expression-2 )
```

#### Parameters

##### **string-expression-1**

The string to be modified by the STUFF function.

##### **start**

The character position at which to begin deleting characters. The first character in the string is position 1.

##### **length**

The number of characters to delete.

##### **string-expression-2**

The string to be inserted. To delete a portion of a string using the STUFF function, use a replacement string of NULL.

#### Returns

LONG BINARY, LONG VARCHAR, or LONG NVARCHAR, depending on the data type of the input expressions.

#### Remarks

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value chocolate pie:

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' );
```

## Related Information

[INSERTSTR Function \[String\] \[page 409\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.113 SUBSTRING Function [String]

Returns a substring of a string.

#### Syntax

```
{ SUBSTRING | SUBSTR }( string-expression , start [ , length ] )
```

## Parameters

### **string-expression**

The string from which a substring is to be returned.

### **start**

The start position of the substring to return, in characters.

### **length**

The length of the substring to return, in characters. If `length` is specified, the substring is restricted to that length.

## Returns

LONG BINARY

LONG VARCHAR

LONG NVARCHAR

UltraLite returns LONG BINARY and LONG VARCHAR

## Remarks

To obtain characters at the end of a string, use the RIGHT function.

If *string-expression* is of binary data type, then the SUBSTRING function behaves as BYTE\_SUBSTR.

This function supports NCHAR inputs and/or outputs. Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics. The behavior of this function depends on the setting of the `ansi_substring` database option. When the `ansi_substring` option is set to On (the default), the behavior of the SUBSTRING function corresponds to ANSI/ISO SQL Standard behavior. The behavior is as follows:

<code>ansi_substring</code> option setting	start value	length value
On	The first character in the string is at position 1. A negative or zero start offset is treated as if the string were padded on the left with non-characters.	A positive <code>length</code> specifies that the substring ends <code>length</code> characters to the right of the starting position.  A negative <code>length</code> returns an error.
Off	The first character in the string is at position 1. A negative starting position specifies a number of characters from the end of the string instead of the beginning.  If <code>start</code> is zero and <code>length</code> is non-negative, a start value of 1 is used. If <code>start</code> is zero and <code>length</code> is negative, a start value of -1 is used.	A positive <code>length</code> specifies that the substring ends <code>length</code> characters to the right of the starting position.  A negative <code>length</code> returns at most <code>length</code> characters up to, and including, the starting position, from the left of the starting position.

**UltraLite:** Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics. In UltraLite, the database does not have an `ansi_substring` option, but the SUBSTR function behaves as if `ansi_substring` is set to on by default. The function's behavior corresponds to ANSI/ISO SQL Standard behavior:

### Start value

The first character in the string is at position 1. A negative or zero start offset is treated as if the string were padded on the left with non-characters.

### Length value

A positive `length` specifies that the substring ends `length` characters to the right of the starting position.

A negative `length` returns an error.

A `length` of zero returns an empty string.

## Standards

### ANSI/ISO SQL Standard

Core Feature. However, the ANSI/ISO SQL Standard implementation differs slightly from the software: in the Standard, SUBSTRING is defined with three parameters using the keywords FROM and FOR, neither of which are required by the software.

## Example

The following table shows the values returned by the SUBSTRING function:

Example	Result
SUBSTRING( 'front yard', 1, 4 )	fron
SUBSTRING( 'back yard', 6, 4 )	yard
SUBSTR( 'abcdefgh', 0, -2 )	Returns an error if the SQL Anywhere ansi_substring option is On
SUBSTR( 'abcdefgh', -2, 2 )	Returns an empty string if the SQL Anywhere ansi_substring option is On

**UltraLite:** The following table shows the values returned by the SUBSTRING function:

Example	Result
SUBSTRING( 'front yard', 1, 4 )	fron
SUBSTRING( 'back yard', 6, 4 )	yard
SUBSTR( 'abcdefgh', 0, -2 )	Returns an error
SUBSTR( 'abcdefgh', -2, 2 )	Returns an empty string

## Related Information

[BYTE\\_SUBSTR Function \[String\] \[page 356\]](#)

[LEFT Function \[String\] \[page 415\]](#)

[RIGHT Function \[String\] \[page 461\]](#)

[CHARINDEX Function \[String\] \[page 364\]](#)

[STRING Function \[String\] \[page 490\]](#)



## 1.13.5.2.114 SUM Function [Aggregate]

Returns the total of the specified expression for each group of rows.

### ☰ Syntax

#### Expression

```
SUM( [ ALL | DISTINCT ] expression )
```

#### Window function

```
SUM( [ ALL ] expression )OVER( window-spec )
```

`window-spec` : see the Remarks section below

#### UltraLite syntax: Expression

```
SUM( [ DISTINCT ] expression )
```

## Parameters

### expression

The name of the expression to be summed. This is commonly a column name.

### [ ALL ] expression

The name of the expression to be summed. This is commonly a column name.

### DISTINCT expression

Computes the sum of the unique values of `expression` within each group.

## Returns

- INTEGER
- DOUBLE
- NUMERIC

## Remarks

Rows where the specified expression is NULL are not included.

Returns NULL for a group containing no rows.

This function can generate an overflow error, resulting in an error being returned. You can use the CAST function on `numeric-expression` to avoid the overflow error.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

## Standards

### ANSI/ISO SQL Standard

Core Feature. When used as a window function, SUM comprises part of optional ANSI/ISO SQL Language Feature T611, "Basic OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional Feature F442, "Mixed column references in set functions". The software does not permit the arguments of an aggregate function to include both a column reference from the query block containing the SUM function, combined with an outer reference.

## Example

The following statement returns the value 3749146.740:

```
SELECT SUM( Salary )  
FROM GROUP0.Employees;
```

## Related Information

[COUNT Function \[Aggregate\] \[page 372\]](#)

[AVG Function \[Aggregate\] \[page 353\]](#)

### 1.13.5.2.115 SWITCHOFFSET Function [Date and Time]

Returns a TIMESTAMP WITH TIME ZONE value that is converted from its original time zone offset to the specified time zone offset.

☞ Syntax

```
SWITCHOFFSET( tmz-expression, time-zone-offset )
```

## Parameters

### **tmz-expression**

The TIMESTAMP WITH TIME ZONE value to be converted.

### **time-zone-offset**

The time zone offset of the result. The value can be an integer representing the minutes before or after Coordinated Universal Time (UTC), a string in the form { + | - } hh:nn, or Z for the Zulu Time Zone. The Zulu Time Zone is the same time zone as UTC.

## Returns

TIMESTAMP WITH TIME ZONE

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following example changes a time zone offset value from -04:00 hours to -07:00 hours. The value returned is 2009-04-03 11:45:12.123-07:00:

```
SELECT CAST ( '2009-04-03 14:45:12.123-04:00' AS datetimeoffset ) AS EDT,  
SWITCHOFFSET ( EDT, '-07:00' ) AS PDT;
```

## 1.13.5.2.116 SYNC\_PROFILE\_OPTION\_VALUE Function [System] - UltraLite

Returns the value of the option corresponding to the given option name.

### ☞ Syntax

```
SYNC_PROFILE_OPTION_VALUE(profile_name, option_name)
```

## Parameters

**profile\_name**

The name of the sync profile to inspect.

**option\_name**

The name of the option to retrieve the corresponding value for.

## Returns

Returns the value of the option corresponding to the given option name.

## Remarks

Option names with periods will retrieve values from a sublist with the given base option name before the period, and the given sublist option name after the period.

## Example

The following statement will return the value of the MobiLinkId option for the synchronization profile named Example:

```
SELECT SYNC_PROFILE_OPTION_VALUE('Example', 'MobiLinkId')
```

## Related Information

[ML\\_GET\\_SERVER\\_NOTIFICATION function \[System\] \[page 438\]](#)

### 1.13.5.2.117 TAN Function [Numeric]

Returns the tangent of a number.

#### ☰ Syntax

```
TAN( numeric-expression )
```

## Parameters

**numeric-expression**

An angle, in radians.

## Returns

DOUBLE

## Remarks

The ATAN and TAN functions are inverse operations.

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

## Standards

**ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following statement returns the value of the tan of 0.52:

```
SELECT TAN( 0.52 );
```

## Related Information

[COS Function \[Numeric\] \[page 370\]](#)

[SIN Function \[Numeric\] \[page 472\]](#)

## 1.13.5.2.118 TODATETIMEOFFSET Function [Date and Time]

Converts a `TIMESTAMP` value to a `TIME STAMP WITH TIME ZONE` value using the specified time zone offset.

### ☞ Syntax

```
TODATETIMEOFFSET( timestamp-expression, time-zone-offset )
```

### Parameters

#### **timestamp-expression**

The `TIMESTAMP` expression to be converted.

#### **time-zone-offset**

The time zone offset. The value can be an `INTEGER` representing minutes before or after UTC, a `VARCHAR` string in the form of { + | - }hh:nn, or the string "Z" for the Zulu Time Zone. The Zulu Time Zone is the same time zone as UTC.

### Returns

`TIMESTAMP WITH TIME ZONE`

### Standards

#### **ANSI/ISO SQL Standard**

Not in the standard.

### Example

The following example converts a `TIMESTAMP` value to a `TIMESTAMP WITH TIME ZONE` value:

```
SELECT CAST('2009-04-03 14:45:12.123' AS TIMESTAMP) AS orig,  
       TODATETIMEOFFSET (orig, '+11:00');
```

## 1.13.5.2.119 TODAY Function [Date and Time]

Returns the current date as a DATE value.

### ☰ Syntax

```
TODAY( [ * ] )
```

### Returns

DATE

### Remarks

TODAY(\*) and TODAY() are semantically equivalent. TODAY is equivalent to the CURRENT DATE special value.

Each instance of the TODAY function in a request is evaluated at most once. Multiple instances of TODAY in the same request may or may not share the identical DATE value.

### Standards

#### ANSI/ISO SQL Standard

Not in the standard.

### Example

The following statements return the current day according to the system clock:

```
SELECT TODAY ( * );  
SELECT CURRENT DATE;
```

## 1.13.5.2.120 TRIM Function [String]

Removes leading and trailing blanks or specified characters from a string.

☞ Syntax

```
TRIM( string-expression [ , trim-char-set ] )
```

### Parameters

#### **string-expression**

The string to be trimmed.

#### **trim-char-set**

The set of characters to trim.

### Returns

VARCHAR, NVARCHAR, LONG VARCHAR, or LONG NVARCHAR, depending on the data type of the input expression.

### Remarks

By default, `trim-char-set` is the space character. You can specify the set of characters to be trimmed.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs and `trim-char-set`.

### Standards

#### **ANSI/ISO SQL Standard**

Core Feature.

The software does not support the additional parameters `trim specification` and `trim character`, as defined in the ANSI/ISO SQL Standard. The implementation of TRIM provided in the software corresponds to a TRIM specification of BOTH.

For the other TRIM specifications defined by the ANSI/ISO SQL Standard (LEADING and TRAILING), the software provides the LTRIM and RTRIM functions respectively.



## Example

The following statement returns the value chocolate with no leading or trailing blanks:

```
SELECT TRIM( '   chocolate   ' );
```

The following statement returns the value def after the specified leading and trailing characters are removed:

```
SELECT TRIM('abccbadeffabcbccba', 'abc' );
```

## Related Information

[LTRIM Function \[String\] \[page 428\]](#)

[RTRIM Function \[String\] \[page 464\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.121 TRUNCNUM Function [Numeric]

Truncates a number at a specified number of places after the decimal point.

### Syntax

```
{ TRUNCNUM | TRUNCATE } ( numeric-expression, integer-expression )
```

## Parameters

### **numeric-expression**

The number to be truncated. This argument may be of type NUMERIC or DOUBLE.

### **integer-expression**

A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative value specifies the number of significant digits to the left of the decimal point at which to round.

## Returns

NUMERIC or DOUBLE

## Remarks

You should use the TRUNCNUM function, not the TRUNCATE function, when truncating numbers.

Use of the TRUNCATE function is not recommended because the word truncate is a keyword, and therefore requires you to either set the quoted\_identifier option to OFF, or put quotes around the word TRUNCATE.

**UltraLite:** If any parameter is NULL, the result is NULL.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 600:

```
SELECT TRUNCNUM( 655, -2 );
```

The following statement: returns the value 655.340:

```
SELECT TRUNCNUM( 655.348, 2 );
```

## Related Information

[ROUND Function \[Numeric\] \[page 463\]](#)

### 1.13.5.2.122 UCASE Function [String]

Converts all characters in a string to uppercase.

☞ Syntax

```
UCASE( string-expression )
```

## Parameters

### **string-expression**

The string to be converted to uppercase.

## Returns

LONG NVARCHAR when used on NCHAR data

LONG VARCHAR when used on CHAR data if the database collation is UCA

Otherwise, the data type is the same as the input data type

UltraLite returns the same data type as the input data type

## Remarks

This function is identical to the UPPER function.

## Standards

### **ANSI/ISO SQL Standard**

Not in the standard. The UPPER function is ANSI/ISO SQL Standard compliant.

## Example

The following statement returns the value CHOCOLATE:

```
SELECT UCASE( 'ChocoLate' );
```

## Related Information

[UPPER Function \[String\] \[page 508\]](#)

[LCASE Function \[String\] \[page 414\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.123 UPPER Function [String]

Converts all characters in a string to uppercase.

☞ Syntax

```
UPPER( string-expression )
```

### Parameters

**string-expression**

The string to be converted to uppercase.

### Returns

LONG NVARCHAR when used on NCHAR data

LONG VARCHAR when used on CHAR data if the database collation is UCA

Otherwise, the data type is the same as the input data type

UltraLite returns the same data type as the input data type

### Remarks

This function is identical to the UCASE function.

### Standards

**ANSI/ISO SQL Standard**

Core feature.

### Example

The following statement returns the value CHOCOLATE:

```
SELECT UPPER( 'ChocoLate' );
```

## Related Information

[UCASE Function \[String\] \[page 506\]](#)

[LCASE Function \[String\] \[page 414\]](#)

[LOWER Function \[String\] \[page 427\]](#)

[STRING Function \[String\] \[page 490\]](#)

### 1.13.5.2.124 UUIDTOSTR Function [String]

Converts a unique identifier value (UUID, also known as GUID) to a string value.

#### ☰ Syntax

```
UUIDTOSTR( uuid-expression )
```

## Parameters

### **uuid-expression**

A unique identifier value.

## Returns

VARCHAR

## Remarks

Converts a unique identifier to a string value in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`, where x is a hexadecimal digit. If the binary value is not a valid uniqueidentifier, NULL is returned.

This function is useful for viewing a UUID value.

#### **i** Note

In databases created before version 9.0.2, the UNIQUEIDENTIFIER data type was defined as a user-defined data type and the STRTOUUID and UUIDTOSTR functions were needed to convert between binary and string representations of UUID values. In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type was changed to a native data type and the database server carries out conversions as needed. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions.

**UltraLite:** In databases created before version 9.0.2, the STRTOUUID and UUIDTOSTR functions were needed to convert between binary and string representations of UUID values. In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type was changed to a native data type. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions..

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement creates a table mytab with two columns. Column pk has a unique identifier data type, and column c1 has an integer data type. It then inserts two rows with the values 1 and 2 respectively into column c1.

```
CREATE TABLE mytab(  
    pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),  
    c1 INT );  
INSERT INTO mytab( c1 ) values ( 1 );  
INSERT INTO mytab( c1 ) values ( 2 );
```

Executing the following SELECT statement returns all the data in the newly created table:

```
SELECT * FROM mytab;
```

You will see a two-column, two-row table. The value displayed for column pk will be binary values.

To convert the unique identifier values into a readable format, execute the following statement:

```
SELECT UUIDTOSTR(pk), c1 FROM mytab;
```

The UUIDTOSTR function is not needed for databases created with version 9.0.2 or later.

## Related Information

[SQL Data Types \[page 288\]](#)

[NEWID Function \[Miscellaneous\] \[page 445\]](#)

[STRTOUUID Function \[String\] \[page 491\]](#)

[STRING Function \[String\] \[page 490\]](#)

## 1.13.5.2.125 WEEKS Function [Date and Time]

Manipulates a `TIMESTAMP` or returns the number of weeks between two `TIMESTAMP` values.

### ☰ Syntax

Returns the number of weeks between 0000-02-29 and a `TIMESTAMP` value

```
WEEKS( timestamp-expression )
```

Returns the number of weeks between two `TIMESTAMP` values

```
WEEKS( timestamp-expression, timestamp-expression )
```

Adds weeks to a `TIMESTAMP` value

```
WEEKS( timestamp-expression, integer-expression )
```

## Parameters

### **timestamp-expression**

A date and time value of type `TIMESTAMP`.

### **integer-expression**

The number of weeks to be added to `timestamp-expression`. If `integer-expression` is negative, the appropriate number of weeks is subtracted from `timestamp-expression`. If you supply an `integer-expression`, `timestamp-expression` must be explicitly cast as a `DATE` or `TIMESTAMP`.

## Returns

`INTEGER` when comparing two `TIMESTAMP` values.

`TIMESTAMP` when adding weeks to a `TIMESTAMP` value.

## Remarks

Given a single date, the `WEEKS` function returns the number of weeks since 0000-02-29.

Given two dates, the `WEEKS` function returns the number of weeks between them. The `WEEKS` function is similar to the `DATEDIFF` function, however the method used to calculate the number of weeks between two dates is not the same and can return a different result. The return value for `WEEKS` is determined by dividing the number of days between the two dates by seven, and then rounding down. However, `DATEDIFF` uses number of week boundaries in its computation. This can cause the values returned from the two functions to be different. For example, if the first date is a Friday and the second date is the following Monday, the `WEEKS`

function returns a difference of 0, but the DATEDIFF function returns a difference of 1. While neither method is better than the other, you should consider the difference when choosing between WEEKS and DATEDIFF.

Given a date and an integer, the WEEKS function adds the integer number of weeks to `timestamp-expression`. With this syntax, you must explicitly cast `timestamp-expression` as a TIME, DATE, or TIMESTAMP data type. If `timestamp-expression` is a TIME value, the current date is assumed.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 8, signifying that 2008-09-13 10:07:12 is eight weeks after 2008-07-13 06:07:12:

```
SELECT WEEKS ( '2008-07-13 06:07:12', '2008-09-13 10:07:12' );
```

The following statement returns the value 104792, signifying that the date is 104792 weeks after 0000-02-29:

```
SELECT WEEKS ( '2008-07-13 06:07:12' );
```

The following statement returns the TIMESTAMP value 2008-06-16 21:05:07.0, indicating the date and time five weeks after 2008-05-12 21:05:07:

```
SELECT WEEKS ( CAST ( '2008-05-12 21:05:07' AS TIMESTAMP ), 5 );
```

## Related Information

[DATEDIFF Function \[Date and Time\] \[page 379\]](#)

[DATEADD Function \[Date and Time\] \[page 377\]](#)

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

## 1.13.5.2.126 YEAR Function [Date and Time]

Returns the year component of the TIMESTAMP argument.

### ☰ Syntax

```
YEAR( timestamp-expression )
```



## Parameters

**timestamp-expression**

A `TIMESTAMP` value.

## Returns

`SMALLINT`

## Remarks

The value returned is the years component of the given `TIMESTAMP` value, returned as a `SMALLINT`.

## Standards

**ANSI/ISO SQL Standard**

Not in the standard.

## Example

The following example returns the value 2001:

```
SELECT YEAR( '2001-09-12' );
```

### 1.13.5.2.127 YEARS Function [Date and Time]

Manipulates a `TIMESTAMP` or returns the number of years between two `TIMESTAMP` values.

#### ☰ Syntax

Return the number of years between year 0000 and a `TIMESTAMP` value

```
YEARS( timestamp-expression )
```

Return the number of years between two `TIMESTAMP` values

```
YEARS( timestamp-expression, timestamp-expression )
```

### Add years to a TIMESTAMP value

```
YEARS( timestamp-expression, integer-expression )
```

## Parameters

### timestamp-expression

A date and time value of type TIMESTAMP.

### integer-expression

The number of years (as a SMALLINT value) to be added to `timestamp-expression`. If `integer-expression` is negative, the appropriate number of years is subtracted from `timestamp-expression`. If you supply an `integer-expression`, the `timestamp-expression` must be explicitly cast as a DATE, TIME, or TIMESTAMP value. If `timestamp-expression` is a TIME, the current year is assumed.

## Returns

SMALLINT when comparing two TIMESTAMP values.

TIMESTAMP when adding years to a TIMESTAMP value.

## Remarks

The value of YEARS is computed by counting the number of first days of the year between the two dates.

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statements both return -4:

```
SELECT YEARS ( '1998-07-13 06:07:12',  
              '1994-03-13 08:07:13' );
```

```
SELECT DATEDIFF( year,
  '1998-07-13 06:07:12',
  '1994-03-13 08:07:13' );
```

The following statements return 1998:

```
SELECT YEARS( '1998-07-13 06:07:12' )
SELECT DATEPART( year, '1998-07-13 06:07:12' );
```

The following statements return the given date advanced 300 years:

```
SELECT YEARS( CAST( '1998-07-13 06:07:12' AS TIMESTAMP ), 300 )
```

```
SELECT DATEADD( year, 300, '1998-07-13 06:07:12' );
```

## Related Information

[DATEDIFF Function \[Date and Time\] \[page 379\]](#)

[DATEADD Function \[Date and Time\] \[page 377\]](#)

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

## 1.13.5.2.128 YMD Function [Date and Time]

Returns a date value corresponding to the given year, month, and day of the month. Arguments are INTEGER values from -32768 to 32767.

### ☰ Syntax

```
YMD( smallint-expression1, smallint-expression2, smallint-expression3 )
```

## Parameters

### **smallint-expression1**

The year.

### **smallint-expression2**

The number of the month. The year is adjusted if the month is outside the range 1-12.

### **smallint-expression3**

The day number. The day can be any integer; the date is adjusted accordingly.

## Returns

DATE

## Standards

### ANSI/ISO SQL Standard

Not in the standard.

## Example

The following statement returns the value 1998-06-12:

```
SELECT YMD( 1998, 06, 12 );
```

If the values are outside their normal range, the date is adjusted accordingly. For example, the following statement returns the DATE value 2000-03-01:

```
SELECT YMD( 1999, 15, 1 );
```

## 1.13.6 UltraLite SQL Statements

The SQL statements supported by UltraLite SQL are a subset of the statements supported by SQL Anywhere databases.

### Before You Begin

- Tables in UltraLite do not support the concept of an owner. As a convenience for existing SQL and for SQL that is programmatically generated, UltraLite still allows the syntax `owner.table-name`. However, the owner is not checked because table owners are not supported in UltraLite.
- UltraLite SQL statement documentation follows the same syntax conventions used by SQL Anywhere statements. Ensure you understand these conventions and how they are used to represent SQL syntax.
- Using UltraLite SQL creates a transaction. A transaction consists of all changes (insert, update, and delete statements) since the last rollback or commit statement. These changes can be made permanent by executing a COMMIT. A ROLLBACK statement causes the changes to be removed.

**In this section:**

### [UltraLite Statement Categories \[page 518\]](#)

SQL statements are organized and identified by the initial keyword in the statement, which is almost always a verb. This action-oriented syntax makes the nature of the language into a set of imperative statements (commands) to the database.

### [ALTER DATABASE SCHEMA FROM FILE Statement \[UltraLite\] \[page 520\]](#)

Modifies the schema definition of an existing UltraLite database using a SQL script.

### [ALTER PUBLICATION Statement \[UltraLite\] \[page 521\]](#)

Alters a publication.

### [ALTER SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 522\]](#)

Alters an UltraLite synchronization profile.

### [ALTER TABLE Statement \[UltraLite\] \[page 524\]](#)

Modifies a table definition.

### [ALTER USER Statement \[UltraLite\] \[page 528\]](#)

Alters user settings.

### [CHECKPOINT Statement \[UltraLite\] \[page 529\]](#)

Checkpoints the database.

### [COMMIT Statement \[UltraLite\] \[page 530\]](#)

Makes the changes to the database permanent.

### [CREATE INDEX Statement \[UltraLite\] \[page 531\]](#)

Creates an index on a specified table.

### [CREATE PUBLICATION Statement \[UltraLite\] \[page 533\]](#)

Creates a publication.

### [CREATE SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 535\]](#)

Creates or replaces an UltraLite synchronization profile.

### [CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

Creates a table.

### [CREATE USER Statement \[UltraLite\] \[page 543\]](#)

Creates a database user or group.

### [DELETE Statement \[UltraLite\] \[page 544\]](#)

Deletes rows from a table in the database.

### [DROP INDEX Statement \[UltraLite\] \[page 545\]](#)

Deletes an index.

### [DROP PUBLICATION Statement \[UltraLite\] \[page 546\]](#)

Deletes publications.

### [DROP SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 547\]](#)

Deletes a synchronization profile.

### [DROP TABLE Statement \[UltraLite\] \[page 548\]](#)

Removes a table, and all its data, from a database.

### [DROP USER Statement \[UltraLite\] \[page 549\]](#)

Drops a user.

### [FROM Clause \[UltraLite\] \[page 550\]](#)

Use this clause to specify the tables or views involved in a SELECT statement.

[INSERT Statement \[UltraLite\] \[page 552\]](#)

Inserts rows into a table.

[LOAD TABLE Statement \[UltraLite\] \[page 553\]](#)

Imports bulk data into a database table from an external file.

[ROLLBACK Statement \[UltraLite\] \[page 557\]](#)

Ends a transaction and reverts any changes made to data since the last COMMIT or ROLLBACK statement was executed.

[SELECT Statement \[UltraLite\] \[page 558\]](#)

Retrieves information from the database.

[SET OPTION Statement \[UltraLite\] \[page 560\]](#)

Changes the values of database options.

[START SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 561\]](#)

Restarts the logging of deleted rows for MobiLink synchronization.

[STOP SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 562\]](#)

Stops the logging of deleted rows for MobiLink synchronization.

[SYNCHRONIZE Statement \[UltraLite\] \[page 563\]](#)

Synchronize an UltraLite database via the MobiLink server.

[TRUNCATE TABLE Statement \[UltraLite\] \[page 565\]](#)

Deletes all rows from a table without deleting the table.

[UNION Statement \[UltraLite\] \[page 566\]](#)

Combines the results of two or more select statements.

[UPDATE Statement \[UltraLite\] \[page 567\]](#)

Modifies rows in a table.

## Related Information

[SQL Statements](#)

[Syntax Conventions](#)

[UltraLite Transaction Processing \[page 585\]](#)

### 1.13.6.1 UltraLite Statement Categories

SQL statements are organized and identified by the initial keyword in the statement, which is almost always a verb. This action-oriented syntax makes the nature of the language into a set of imperative statements (commands) to the database.

In UltraLite, supported SQL statements can be classified as follows:

#### **Data retrieval statements**

Also known as queries. These statements allow select rows of data expressions from tables. Data retrieval is achieved with the SELECT statement.

#### **Data manipulation statements**

Allow you to change content in the database. Data manipulation is achieved with the following statements:

- INSERT
- UPDATE
- DELETE

#### **Data definition statements**

Allow you to define the structure or schema of a database. The schema can be changed with the following statements:

- ALTER DATABASE SCHEMA FROM FILE
- CREATE INDEX
- CREATE TABLE
- DROP INDEX
- DROP TABLE
- ALTER TABLE
- TRUNCATE TABLE

#### **Transaction control statements**

Allow you to control transactions within your UltraLite application. Transaction control is achieved with the following statements:

- CHECKPOINT
- COMMIT
- ROLLBACK

#### **Synchronization management**

Allow you to temporarily control synchronization with a MobiLink server. Synchronization management is achieved with:

- START SYNCHRONIZATION DELETE
- STOP SYNCHRONIZATION DELETE
- CREATE PUBLICATION
- ALTER PUBLICATION
- DROP PUBLICATION

## **Related Information**

[Operators in UltraLite \[page 285\]](#)

[Expressions in UltraLite \[page 264\]](#)

[SELECT Statement \[UltraLite\] \[page 558\]](#)

[INSERT Statement \[UltraLite\] \[page 552\]](#)

[UPDATE Statement \[UltraLite\] \[page 567\]](#)

[DELETE Statement \[UltraLite\] \[page 544\]](#)

[ALTER DATABASE SCHEMA FROM FILE Statement \[UltraLite\] \[page 520\]](#)

[CREATE INDEX Statement \[UltraLite\] \[page 531\]](#)

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

[DROP INDEX Statement \[UltraLite\] \[page 545\]](#)  
[DROP TABLE Statement \[UltraLite\] \[page 548\]](#)  
[ALTER TABLE Statement \[UltraLite\] \[page 524\]](#)  
[TRUNCATE TABLE Statement \[UltraLite\] \[page 565\]](#)  
[CHECKPOINT Statement \[UltraLite\] \[page 529\]](#)  
[COMMIT Statement \[UltraLite\] \[page 530\]](#)  
[ROLLBACK Statement \[UltraLite\] \[page 557\]](#)  
[START SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 561\]](#)  
[STOP SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 562\]](#)  
[CREATE PUBLICATION Statement \[UltraLite\] \[page 533\]](#)  
[ALTER PUBLICATION Statement \[UltraLite\] \[page 521\]](#)  
[DROP PUBLICATION Statement \[UltraLite\] \[page 546\]](#)

## 1.13.6.2 ALTER DATABASE SCHEMA FROM FILE Statement [UltraLite]

Modifies the schema definition of an existing UltraLite database using a SQL script.

≡ Syntax

```
ALTER DATABASE SCHEMA FROM FILE filename
```

### Parameters

**filename**

Defines the name and path to the SQL script used to upgrade the schema of an existing UltraLite database.

### Remarks

Use either `ulinit` or `ulunload` to extract the DDL statements required for your script. By using these utilities, you ensure that the DDL statements are syntactically correct. Use `ulinit` (`-l logfile` option) or `ulunload` (using the `-n -s output-file` options).

Backup the database before executing this statement.

The character set of the SQL script file must match the character set of the database you want to upgrade.

Ensure that your device is not reset while this statement is executing. If you reset the device during a schema upgrade, the UltraLite database becomes unusable.

Any rows that do not fit into the schema will be dropped (for instance if a uniqueness constraint is added and multiple rows contain the same values, all but one row will be dropped). In this case, the



SQL\_ROW\_DROPPED\_DURING\_SCHEMA\_UPGRADE warning is generated. You can use this warning to detect the error and restore the database from the backup version.

## Example

The following statement modifies the schema of the database using a SQL script, `MySchema.sql`:

```
ALTER DATABASE SCHEMA FROM FILE 'MySchema.sql'
```

## Related Information

[UltraLite Database Schemas \[page 52\]](#)

[Deploying UltraLite Database Schema Upgrades \[page 131\]](#)

[UltraLite Initialize Database Utility \(ulinit\) \[page 227\]](#)

[UltraLite Database Unload Utility \(ulunload\) \[page 243\]](#)

### 1.13.6.3 ALTER PUBLICATION Statement [UltraLite]

Alters a publication.

#### ⌵ Syntax

```
ALTER PUBLICATION publication-name alterpub-clause
```

```
alterpub-clause :  
  ADD TABLE table-name [ WHERE search-condition ]  
 | ALTER TABLE table-name [ WHERE search-condition ]  
 | { DROP | DELETE } TABLE table-name  
 | RENAME publication-name
```

## Remarks

A publication identifies data in a remote database that is to be synchronized.

## Side effects

Automatic commit.

## Example

The following ALTER PUBLICATION statement adds the Customers table to the pub\_contact publication.

```
ALTER PUBLICATION pub_contact
ADD TABLE Customers
```

## Related Information

[UltraLite Client Synchronization Design \[page 79\]](#)

[Search Conditions in UltraLite \[page 272\]](#)

[CREATE PUBLICATION Statement \[UltraLite\] \[page 533\]](#)

[DROP PUBLICATION Statement \[UltraLite\] \[page 546\]](#)

[START SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 561\]](#)

[STOP SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 562\]](#)

## 1.13.6.4 ALTER SYNCHRONIZATION PROFILE Statement [UltraLite]

Alters an UltraLite synchronization profile.

### Syntax

```
ALTER SYNCHRONIZATION PROFILE sync-profile-name
MERGE sync-option [; ... ]
```

```
sync-option :
sync-option-name = sync-option-value
```

```
sync-option-name : string
```

```
sync-option-value : string
```

## Parameters

### sync-profile-name

The name of the synchronization profile.

### MERGE clause

Use this clause to change existing, or add new, options to a synchronization profile.

**sync-option**

A string of one or more synchronization option value pairs, separated by semicolons. For example, 'option1=value1;option2=value2'.

**sync-option-name**

The name of the synchronization profile option.

**sync-option-value**

The value for the synchronization profile option.

## Remarks

Synchronization profiles define how an UltraLite database synchronizes with the MobiLink server.

You can use the MERGE clause to make changes to an existing synchronization profile. When using this clause, only the synchronization options that are specified in the MERGE clause are changed. To remove a synchronization option from a synchronization profile, the sync-option string should look like 'option1=' (to set the option to an empty value).

The STREAM synchronization profile option is different from the other options because its value contains a sub-list. For example: 'STREAM=TCPIP{host=192.168.1.1;port=1234}'. In this case 'host=192.168.1.1;port=1234' is the sub-list. To add or remove a sub-list value, use a period between the STREAM sync-option-name and the sub-option-name. For example, MERGE 'stream.port=5678;stream.host=;compression=zlib' results in a synchronization profile of: stream=TCPIP{port=5678;compression=zlib}. Attempting to set the stream to a new value will replace the entire stream value. For example: MERGE 'stream=HTTPS' results in a synchronization profile of: stream=HTTPS{}

## Side effects

None.

## Example

The following is an example of the ALTER SYNCHRONIZATION PROFILE...REPLACE statement:

```
CREATE SYNCHRONIZATION PROFILE myProfile1;  
ALTER SYNCHRONIZATION PROFILE myProfile1  
  REPLACE 'publications=p1;uploadonly=on';
```

The following is an example of the ALTER SYNCHRONIZATION PROFILE...MERGE statement.

```
CREATE SYNCHRONIZATION PROFILE myProfile2 'publications=p1;  
ALTER SYNCHRONIZATION PROFILE myProfile2  
  MERGE 'publications=p2;uploadonly=on';
```

The following examples illustrate the changes that occur after executing a sequence of ALTER SYNCHRONIZATION PROFILE commands with different options.

Suppose `myProfile1='MobiLinkUID=mary;ScriptVersion=default'`.

After executing `ALTER SYNCHRONIZATION PROFILE myProfile1 REPLACE 'MobiLinkPwd=sql;ScriptVersion=1'`, `myProfile1` is `'MobiLinkPwd=sql;ScriptVersion=1'`.

After executing `ALTER SYNCHRONIZATION PROFILE myProfile1 MERGE 'MobiLinkUID=mary;STREAM=tcPIP'`, `myProfile1` is `'MobiLinkPwd=sql;ScriptVersion=1;MobiLinkUID=mary;STREAM=tcPIP'`.

After executing `ALTER SYNCHRONIZATION PROFILE myProfile1 MERGE 'MobiLinkUID=;STREAM.host=192.168.1.1;STREAM.port=1234;ScriptVersion=;'`, `myProfile1` is `'MobiLinkPwd=sql;STREAM=tcPIP{192.168.1.1;port=1234}'`.

After executing `ALTER SYNCHRONIZATION PROFILE myProfile1 MERGE 'MobiLinkPwd=;Ping=yes;STREAM=HTTP'`, `myProfile1` is `'Ping=yes;STREAM=HTTP'`.

After executing `ALTER SYNCHRONIZATION PROFILE myProfile1 MERGE 'STREAM=HTTP{host=192.168.1.1}'`, `myProfile1` is `'Ping=yes;STREAM=HTTP{host=192.168.1.1}'`.

## Related Information

[UltraLite Synchronization Profile Options \[page 241\]](#)

[DROP SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 547\]](#)

[SYNCHRONIZE Statement \[UltraLite\] \[page 563\]](#)

### 1.13.6.5 ALTER TABLE Statement [UltraLite]

Modifies a table definition.

#### ⌵ Syntax

```
ALTER TABLE table-name {  
  add-clause  
  | modify-clause  
  | drop-clause  
  | rename-clause  
}
```

```
add-clause :  
  ADD { column-definition | table-constraint }
```

```
modify-clause :  
  ALTER column-definition | sync-constraint
```

```
drop-clause :  
  DROP { column-name | CONSTRAINT constraint-name }
```

```

rename-clause :
RENAME {
  new-table-name
  | [ old-column-name TO ] new-column-name
  | CONSTRAINT old-constraint-name TO new-constraint-name }

```

```

column-definition :
column-name data-type
  [ [ NOT ] NULL ]
  [ DEFAULT column-default ]
  [ UNIQUE ]

```

```

column-default :
GLOBAL AUTOINCREMENT [ ( number ) ]
| AUTOINCREMENT
| CURRENT DATE
| CURRENT TIME
| CURRENT TIMESTAMP
| NULL
| NEWID()
| constant-value

```

```

table-constraint :
[ CONSTRAINT constraint-name ]
{ fkey-constraint | unique-key-constraint }
[ WITH MAX HASH SIZE integer ]

```

```

fkey-constraint :
[ NOT NULL ] FOREIGN KEY [ role-name ] ( ordered-column-list )
  REFERENCES table-name ( column-name, ... )
  [ CHECK ON COMMIT ]

```

```

unique-key-constraint :
UNIQUE ( ordered-column-list )

```

```

ordered-column-list :
( column-name [ ASC | DESC ], ... )

```

```

sync-constraint :SYNCHRONIZE {ON| OFF|ALL|DOWNLOAD}

```

## Parameters

### add-clause

Adds a new column or table constraint to the table:

Adds a new column or table constraint to the table:

#### ADD column-definition clause

Adds a new column to the table. If the column has a default value, all rows in the new column are populated with that default value.

#### ADD table-constraint clause

Adds a constraint to the table. The optional constraint name allows you to modify or drop individual constraints at a later time, rather than having to modify the entire table constraint.

When adding a new unique constraint, all constraint columns must be non nullable. To add a unique constraint, alter the column to be NOT NULL.

#### **i Note**

You cannot add a primary key in UltraLite.

#### **modify-clause**

Change a single column definition. You cannot use primary keys in the `column-definition` when part of an ALTER statement. If necessary, the data in the modified column is converted to the new data type. If a conversion error occurs, the operation will fail and the table is left unchanged.

#### **drop-clause**

Delete a column or a table constraint:

##### **DROP column-name**

Delete the column from the table. If the column is contained in any index, uniqueness constraint, foreign key, or primary key, then the object must be deleted *before* UltraLite can delete the column.

##### **DROP CONSTRAINT table-constraint**

Delete the named constraint from the table definition.

#### **i Note**

You cannot drop a primary key in UltraLite.

#### **rename-clause**

Change the name of a table, column, or constraint:

##### **RENAME new-table-name**

Change the name of the table to `new-table-name`. Any applications using the old table name must be modified. Foreign keys that were automatically assigned the old table name will not change names.

##### **RENAME old-column-name TO new-column-name**

Change the name of the column to the `new-column-name`. Any applications using the old column name will need to be modified.

##### **RENAME old-constraint-name TO new-constraint-name**

Change the name of the constraint to the `new-constraint-name`. Any applications using the old constraint name need to be modified.

#### **i Note**

You cannot rename a primary key in UltraLite.

#### **column-constraint**

A column constraint restricts the values the column can hold to ensure the integrity of data in the database. A column constraint can only be UNIQUE.

#### **UNIQUE**

Identifies one or more columns that uniquely identify each row in the table. No two rows in the table can have the same values in all the named column(s). A table may have more than one unique constraint.

#### **sync-constraint clause**

Specify a sync constraint to determine whether a table can be synchronized or not and whether all rows are uploaded, just changes to the table are uploaded, or no changes to the table are uploaded.

#### **SYNCHRONIZE ON**

Default setting - the table can be synchronized and only changes to the table are sent in the upload.

#### **SYNCHRONIZE OFF**

The table cannot be synchronized and it is an error to include the table in a publication.

#### **SYNCHRONIZE ALL**

The table can be synchronized and all rows in the table are sent in the upload.

#### **SYNCHRONIZE DOWNLOAD**

The table can be synchronized with changes to the consolidated database but no local changes are uploaded.

## Remarks

Only one `table-constraint` or `column-constraint` can be added, modified, or deleted in one ALTER TABLE statement.

The role name is the name of the foreign key. The main function of the `role-name` is to distinguish two foreign keys to the same table. Alternatively, you can name the foreign key with CONSTRAINT `constraint-name`. However, do not use both methods to name a foreign key.

You cannot MODIFY a table or column constraint. To change a constraint, you must DELETE the old constraint and ADD the new constraint.

ALTER TABLE cannot execute if a statement that affects the table is already being referenced by another request or query. Similarly, UltraLite does not process requests referencing the table while that table is being altered. Furthermore, you cannot execute ALTER TABLE when the database includes active queries or uncommitted transactions.

For UltraLite.NET users: You cannot execute this statement unless you also call the `ULBulkCopy.Dispose` method for all data objects (for example, `ULDataReader`).

Statements are not released if schema changes are initiated at the same time.

## Example

The following statement drops the Street column from a fictitious table called MyEmployees.

```
ALTER TABLE MyEmployees
DROP Street
```

The following example changes the Street column of the fictitious table, MyCustomers, to hold approximately 50 characters.

```
ALTER TABLE MyCustomers
ALTER Street CHAR(50)
```

## Related Information

[UltraLite Database Schemas \[page 52\]](#)

[SQL Data Types \[page 288\]](#)

[Table Alteration](#)

[Table and Column Constraints](#)

[Partition Sizes \[page 77\]](#)

[Methods for Finding the Last Assigned GLOBAL AUTOINCREMENT Value \[page 77\]](#)

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

[DROP TABLE Statement \[UltraLite\] \[page 548\]](#)

### 1.13.6.6 ALTER USER Statement [UltraLite]

Alters user settings.

#### Syntax 1

```
ALTER USER user-name [ IDENTIFIED BY password ]
```

#### Parameters

**user-name**

The name of the user.

**IDENTIFIED BY clause**

The password for the user.



## Remarks

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

- User IDs cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons
- Passwords are case-sensitive and they cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons
  - be longer than 255 bytes in length

## Side effects

None.

## Example

The following alters a user named SQLTester. The password is set to "welcome".

```
ALTER USER SQLTester IDENTIFIED BY welcome
```

## 1.13.6.7 CHECKPOINT Statement [UltraLite]

Checkpoint the database.

☰ Syntax

*CHECKPOINT*

## Remarks

You can use the CHECKPOINT statement as a trigger for a commit flush. A commit flush writes uncommitted transactions to storage.

If you are using the Embedded SQL API, you can also use the ULCheckpoint method. If you are writing a C++ component application, you can also use the Checkpoint method on a connection object. All other APIs must use this statement.

## Side effects

While this statement flushes any pending committed transactions to storage, it does not commit or flush current transactions.

## Example

The following statement performs a checkpoint of the database:

```
CHECKPOINT
```

## Related Information

[Flush Single or Grouped Transactions \[page 586\]](#)

[COMMIT Statement \[UltraLite\] \[page 530\]](#)

[UltraLite COMMIT\\_FLUSH Connection Parameter \[page 188\]](#)

## 1.13.6.8 COMMIT Statement [UltraLite]

Makes the changes to the database permanent.

☞ Syntax

```
COMMIT [ WORK ]
```

## Remarks

Using UltraLite SQL creates a transaction. A transaction consists of all changes (INSERTs, UPDATEs, and DELETEs) since the last ROLLBACK or COMMIT. The COMMIT statement ends the current transaction and makes all changes made during the transaction permanent in the database.

Changes to the database objects using the ALTER, CREATE, and DROP statements are committed automatically.

## Example

The following statement makes the changes in the current transaction permanent in the database:

```
COMMIT
```

## Related Information

[CHECKPOINT Statement \[UltraLite\] \[page 529\]](#)

[ROLLBACK Statement \[UltraLite\] \[page 557\]](#)

### 1.13.6.9 CREATE INDEX Statement [UltraLite]

Creates an index on a specified table.

#### Syntax

```
CREATE [ UNIQUE ] INDEX [ IF NOT EXISTS ] [ index-name ]  
ON table-name ( ordered-column-list )  
[ WITH MAX HASH SIZE integer ]
```

```
ordered-column-list :  
( column-name [ ASC | DESC ], ... )
```

## Parameters

### UNIQUE

The UNIQUE attribute ensures that there are not two rows in the table with identical values in all the columns in the index. Each index key must be unique or contain a NULL in at least one column.

There is a difference between a unique constraint on a table and a unique index. Columns of a unique index are allowed to be NULL, while columns in a unique constraint are not. A foreign key can reference either a primary key or a unique constraint, but not a unique index, because it can include multiple instances of NULL.

If the columns in a unique constraint are changed during an update, and a foreign key references that unique constraint, any rows no longer referencing rows in the unique constraint are deleted from the remote.

### IF NOT EXISTS clause

When the IF NOT EXISTS attribute is specified and the named index already exists, no changes are made and an error is not returned.

### ordered-column-list

An ordered list of columns. Column values in the index can be sorted in ascending or descending order.

#### **WITH MAX HASH SIZE**

Sets the hash size (in bytes) for this index. This value overrides the default MaxHashSize database property in effect for the database.

## **Remarks**

UltraLite automatically creates indexes for primary keys and for unique constraints.

Indexes can improve query performance by providing quick ways for UltraLite to look up specific rows. Conversely, because they have to be maintained, indexes may slow down synchronization and INSERT, DELETE, and UPDATE statements.

Indexes are automatically used to improve the performance of queries issued to the database, and to sort queries with an ORDER BY clause. Once an index is created, it is never referenced in a SQL statement again except to remove it with DROP INDEX.

Indexes use space in the database. Also, the additional work required to maintain indexes can affect the performance of data modification operations. For these reasons, you should avoid creating indexes that do not improve query performance.

UltraLite does not process requests or queries referencing the index while the CREATE INDEX statement is being processed. Furthermore, you cannot execute CREATE INDEX when the database includes active queries or uncommitted transactions.

Use execution plans to optimize queries.

For UltraLite.NET users, you cannot execute this statement unless you also call the ULBulkCopy.Dispose method for all data objects (for example, ULDataReader).

Statements are not released if database schema changes are initiated at the same time.

## **Side effects**

- Automatic commit.

## **Example**

The following statement creates a two-column index on the Employees table.

```
CREATE INDEX employee_name_index
ON Employees ( Surname, GivenName )
```

The following statement creates an index on the SalesOrderItems table for the ProductID column.

```
CREATE INDEX item_prod
```

```
ON SalesOrderItems ( ProductID )
```

The following scenario illustrates the effects of MAX HASH SIZE on an UltraLite Java edition database, given an Employees table that contains an Initials column that is VARCHAR( 3 ) and an EmployeeID column that is TINY.

The following statement completely hashes all values when only ASCII7 characters are used:

```
CREATE INDEX ascii_a ON Employees( Initials ) WITH MAX HASH SIZE 3
```

The following statement completely hashes all values no matter what characters they contain:

```
CREATE INDEX unicode_a ON Employees( Initials ) WITH MAX HASH SIZE 9
```

The following statement only hashes the Initials values even when only ASCII characters are used because the first 9 bytes for Initials are reserved:

```
CREATE INDEX compound_1 ON Employees( Initials, EmployeeID ) WITH MAX HASH SIZE 9
```

The following statement completely hashes both Initials and EmployeeID values:

```
CREATE INDEX compound_2 ON Employees( Initials, EmployeeID ) WITH MAX HASH SIZE  
10
```

## Related Information

[UltraLite Performance Tips \[page 569\]](#)

[UltraLite Indexes \[page 62\]](#)

[UltraLite Database Schemas \[page 52\]](#)

[Execution Plans in UltraLite \[page 575\]](#)

[Reading Database Properties \[page 43\]](#)

[DROP INDEX Statement \[UltraLite\] \[page 545\]](#)

[UltraLite max\\_hash\\_size Creation Option \[page 161\]](#)

### 1.13.6.10 CREATE PUBLICATION Statement [UltraLite]

Creates a publication.

#### ☰ Syntax

```
CREATE PUBLICATION [ IF NOT EXISTS ] publication-name  
( TABLE table-name [ WHERE search-condition ], ... )
```

## Parameters

### IF NOT EXISTS clause

When the IF NOT EXISTS clause is specified and the named publication already exists, no changes are made and an error is not returned.

### TABLE clause

Use the table to include a TABLE in the publication. There is no limit to the number of TABLE clauses.

### WHERE clause

If a WHERE clause is specified, only rows satisfying *search-condition* are considered for upload from the associated table during synchronization.

If you do not specify a WHERE clause, every row in the table that has changed in UltraLite since the last synchronization is considered for upload.

## Remarks

A publication identifies synchronized data in an UltraLite remote database.

A publication establishes tables that are synchronized during a single synchronization operation, and determines which data is uploaded to the MobiLink server. The MobiLink server may send back rows for these (and only these) tables during its download session; however, rows that are downloaded do not have to satisfy the WHERE clause for a table.

Only entire tables can be published. You cannot publish specific columns of a table in UltraLite.

## Side effects

- Automatic commit.

## Example

The following statement publishes all the columns and rows of two tables.

```
CREATE PUBLICATION pub_contact (  
  TABLE Contacts,  
  TABLE Customers  
)
```

The following statement publishes only the rows of the Customers table where the State column contains MN.

```
CREATE PUBLICATION pub_customer (  
  TABLE Customers  
  WHERE State = 'MN'  
)
```

## Related Information

[UltraLite Clients \[page 73\]](#)

[Search Conditions in UltraLite \[page 272\]](#)

[DROP PUBLICATION Statement \[UltraLite\] \[page 546\]](#)

[ALTER PUBLICATION Statement \[UltraLite\] \[page 521\]](#)

[Search Conditions in UltraLite \[page 272\]](#)

### 1.13.6.11 CREATE SYNCHRONIZATION PROFILE Statement [UltraLite]

Creates or replaces an UltraLite synchronization profile.

#### ≡ Syntax

```
CREATE [OR REPLACE] SYNCHRONIZATION PROFILE sync-profile-name sync-option [;...]
```

```
sync-option :  
sync-option-name = sync-option-value
```

```
sync-option-name : string
```

```
sync-option-value : string
```

## Parameters

### OR REPLACE clause

If the named synchronization profile already exists, then it will be replaced. If the profile does not exist, it will be created.

### sync-profile-name

The name of the synchronization profile.

### sync-option

A string of one or more synchronization option value pairs, separated by semicolons. For example, 'option1=value1;option2=value2'.

### sync-option-name

The name of the synchronization profile option.

### sync-option-value

The value for the synchronization profile option.

## Remarks

Synchronization profiles define how an UltraLite database synchronizes with the MobiLink server.

You can use the REPLACE clause to make changes to an existing synchronization profile. This clause replaces the contents of the synchronization profile with whatever is contained in the new sync-option string. This approach is the same as dropping the synchronization profile and then creating one with the same name but using the new string. Therefore, a synchronization profile does not need to contain a full synchronization definition because parameters can be merged in or overridden at synchronization time.

The STREAM synchronization profile option is different from the other options because its value contains a sub-list. For example: 'STREAM=TCPIP{host=192.168.1.1;port=1234}'. In this case 'host=192.168.1.1;port=1234' is the sub-list. To add or remove a sub-list value, use a period between the STREAM sync-option-name and the sub-option-name. For example, MERGE 'stream.port=5678;stream.host=;compression=zlib' results in a synchronization profile of: stream=TCPIP{port=5678;compression=zlib}. Attempting to set the stream to a new value will replace the entire stream value. For example: MERGE 'stream=HTTPS' results in a synchronization profile of: stream=HTTPS{}

## Side effects

None.

## Example

The following creates a synchronization profile called Test1.

```
CREATE SYNCHRONIZATION PROFILE Test1
'MobiLinkUid=mary;Stream=TCPIP{host=192.168.1.1;port=1234}'
```

## Related Information

[UltraLite Synchronization Profile Options \[page 241\]](#)

[ALTER SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 522\]](#)

[DROP SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 547\]](#)

[SYNCHRONIZE Statement \[UltraLite\] \[page 563\]](#)



## 1.13.6.12 CREATE TABLE Statement [UltraLite]

Creates a table.

### Syntax

```
CREATE TABLE [ IF NOT EXISTS ] table-name (  
  { column-definition | table-constraint | sync-constraint }, ...  
)
```

```
column-definition :  
column-name data-type  
[ [ NOT ] NULL ]  
[ DEFAULT column-default ]  
[ STORE AS FILE (file-name-column) [ CASCADE DELETE ]  
[ column-constraint ]
```

```
column-default :  
AUTOFILENAME (prefix, extension)  
| GLOBAL AUTOINCREMENT [ ( number ) ]  
| AUTOINCREMENT  
| CURRENT DATE  
| CURRENT TIME  
| CURRENT TIMESTAMP  
| CURRENT UTC TIMESTAMP  
| NULL  
| NEWID()  
| constant-value
```

```
file-name  
"filename"
```

```
column-constraint :  
PRIMARY KEY  
| UNIQUE
```

```
table-constraint :  
{ [ CONSTRAINT constraint-name ]  
  pkey-constraint  
  | fkey-constraint  
  | unique-key-constraint }  
[ WITH MAX HASH SIZE integer ]
```

```
pkey-constraint :  
PRIMARY KEY [ ordered-column-list ]
```

```
fkey-constraint :  
[ NOT NULL ] FOREIGN KEY [ role-name ] ( ordered-column-list )  
  REFERENCES table-name ( column-name, ... )  
  [ CHECK ON COMMIT ]
```

```
unique-key-constraint :  
UNIQUE ( ordered-column-list )
```

```
ordered-column-list :  
( column-name [ ASC | DESC ], ... )
```

```
sync-constraint :SYNCHRONIZE {ON | OFF | ALL | DOWNLOAD}
```

## Parameters

### IF NOT EXISTS clause

Use this clause to create a table. No changes are made if the named table already exists, and an error is not returned.

### column-definition

Defines a column in a table. Available parameters for this clause include:

#### column-name

The column name is an identifier. Two columns in the same table cannot have the same name.

#### data-type

The data type of the column.

#### [ NOT ] NULL

If NOT NULL is specified, or if the column is in a PRIMARY KEY or UNIQUE constraint, the column cannot contain NULL in any row. Otherwise, NULL is allowed.

#### column-default

Sets the default value for the column. If a DEFAULT value is specified, it is used as the value for the column in any INSERT statement that does not specify a value for the column. If no DEFAULT is specified, it is equivalent to DEFAULT NULL. Default options include:

#### AUTOFILENAME

This clause supports the storing of external BLOB files in a partitioned UltraLite Java edition database.

When partitioning the database, the column designated to store the file names requires the AUTOFILENAME(prefix,extension) clause. This clause specifies how new filenames are to be generated for downloaded BLOB values. The prefix and extension values are string literal constants.

#### AUTOINCREMENT

When using AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type. On inserts into the table, if a value is not specified for the AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column that is larger than the current maximum value for the column, that value is used as a starting point for subsequent inserts.

#### i Note

In UltraLite, the autoincrement value is not set to 0 when the table is created, and AUTOINCREMENT generates negative numbers when a signed data type is used for the column. Therefore, declare AUTOINCREMENT columns as unsigned integers to prevent negative values from being used.

#### GLOBAL AUTOINCREMENT

Similar to AUTOINCREMENT, except that the domain is partitioned. Each partition contains the same number of values. You assign each copy of the database a unique global database identification number. UltraLite supplies default values in a database only from the partition uniquely identified by that database's number.

### **i Note**

If the column is of type BIGINT or UNSIGNED BIGINT, the default partition size is  $2^{32} = 4294967296$ ; for columns of all other types, the default partition size is  $2^{16} = 65536$ . Since these defaults may be inappropriate, especially if your column is not of type INT or BIGINT, it is best to specify the partition size explicitly.

### **[ NOT ] NULL**

Controls whether the column can contain NULLs.

### **NEWID( )**

A function that generates a unique identifier value.

### **CURRENT TIMESTAMP**

Combines CURRENT DATE and CURRENT TIME to form a TIMESTAMP value containing the year, month, day, hour, minute, second, and fraction of a second. The fraction of a second is stored to 3 decimal places. The accuracy is limited by the accuracy of the system clock.

### **CURRENT UTC TIMESTAMP**

A TIMESTAMP WITH TIME ZONE value containing the Coordinated Universal Time (UTC) comprised of the year, month, day, hour, minute, second, fraction of a second, and time zone. The fraction of a second is stored to 3 decimal places. The accuracy is limited by the accuracy of the system clock.

### **CURRENT DATE**

Stores the current year, month, and day.

### **CURRENT TIME**

Stores the current hour, minute, second and fraction of a second.

### **constant-value**

A constant for the data type of the column. Typically the constant is a number or a string.

### **STORE AS FILE (file-name-column) [CASCADE DELETE]**

Supported by UltraLite Java edition databases only.

Specify that a LONG BINARY column is to be stored externally (partitioning the database) and specify the `file-name-column` to name the column that will be used to store the file names of the externally stored BLOB values. A column with this clause must be of type LONG BINARY and behave as a read-only column.

### **column-constraint clause**

Specify a column constraint to restrict the values allowed in a column. A column constraint can be one of:

#### **PRIMARY KEY**

When set as part of a `column-constraint`, the PRIMARY KEY clause sets the column as the primary key for the table. Primary keys uniquely identify each row in a table. By default, columns included in primary keys do not allow NULL.

## UNIQUE

Identifies one or more columns that uniquely identify each row in the table. No two rows in the table can have the same values in all the named column(s). A table may have more than one unique constraint. NULL values are not allowed.

### table-constraint clause

Specify a table constraint to restrict the values that one or more columns in the table can hold. Use the CONSTRAINT clause to specify an identifier for the table constraint. Table constraints can be in the form of a primary key constraint, a foreign key constraint, or a unique constraint, as defined below:

#### pkey-constraint clause

Sets the specified column(s) as the primary key for the table. Primary keys uniquely identify each row in a table. Columns included in primary keys cannot allow NULLs.

#### fkey-constraint clause

Specify a foreign key constraint to restrict values for one or more columns that must match the values in a primary key (or a unique constraint) of another table.

#### NOT NULL clause

Specify NOT NULL to disallow NULLs in the foreign key columns. A NULL in a foreign key means that no row in the primary table corresponds to this row in the foreign table. If at least one value in a multi-column foreign key is NULL, there is no restriction on the values that can be held in other columns of the key.

#### role-name clause

Specify a `role-name` to name the foreign key. `role-name` is used to distinguish foreign keys within the same table. Alternatively, you can name the foreign key using CONSTRAINT `constraint-name`. However, do not use both methods to name a foreign key.

#### REFERENCES clause

Specify the REFERENCES clause to define one or more columns in the primary table to use as the foreign key constraint. Any `column-name` you specify in a REFERENCES column constraint must be a column in the primary table, and must be subject to a unique constraint or primary key constraint.

#### CHECK ON COMMIT

not supported by UltraLite Java edition databases. Specify CHECK ON COMMIT to cause the database server to wait for a COMMIT before enforcing foreign key constraints. By default, foreign key constraints are enforced immediately during insert, update, or delete operations. However, when CHECK ON COMMIT is set, database changes can be made in any order, even if they violate foreign key constraints, if inconsistent data is resolved before the next COMMIT.

#### unique-key-constraint clause

Specify a unique constraint to identify one or more columns that uniquely identify each row in the table. No two rows in the table can have the same values in all the named column(s). A table may have more than one unique constraint.

#### WITH MAX HASH SIZE

Sets the hash size (in bytes) for this index. This value overrides the default MaxHashSize database property in effect for the database.

### sync-constraint clause

Specify a sync constraint to determine whether a table can be synchronized or not and whether all rows are uploaded, just changes to the table are uploaded, or no changes to the table are uploaded.

#### **SYNCHRONIZE ON**

Default setting - the table can be synchronized and only changes to the table are sent in the upload.

#### **SYNCHRONIZE OFF**

The table cannot be synchronized and it is an error to include the table in a publication.

#### **SYNCHRONIZE ALL**

The table can be synchronized and all rows in the table are sent in the upload. This constraint is not supported by UltraLite Java edition databases.

#### **SYNCHRONIZE DOWNLOAD**

The table can be synchronized with changes to the consolidated database but no local changes are uploaded.

## **Remarks**

Column constraints are normally used unless the constraint references more than one column in the table. In these cases, a table constraint must be used. If a statement causes a violation of a constraint, execution of the statement does not complete. Any changes made by the statement before error detection are undone, and an error is reported.

Each row in the table has a unique primary key value.

If no role name is specified, the role name is assigned as follows:

1. If there is no foreign key with a role name the same as the table name, the table name is assigned as the role name.
2. If the table name is already taken, the role name is the table name concatenated with a zero-padded, three-digit number unique to the table.

#### **Schema changes**

Statements are not released if database schema changes are initiated at the same time.

UltraLite does not process requests or queries referencing the table while the CREATE TABLE statement is being processed. Furthermore, you cannot execute CREATE TABLE when the database includes active queries or uncommitted transactions.

For UltraLite.NET users: You cannot execute this statement unless you also call the ULBulkCopy.Dispose method for all data objects (for example, ULDataReader).

## **Side effects**

Automatic commit.

## Example

The following statement creates a table for a library database to hold book information.

```
CREATE TABLE library_books (  
  isbn CHAR(20) PRIMARY KEY,  
  copyright_date DATE,  
  title CHAR(100),  
  author CHAR(50),  
  location CHAR(50),  
  FOREIGN KEY location REFERENCES room  
)
```

The following statement creates a table for a library database to hold information about borrowed books. The default value for date\_borrowed indicates that the book is borrowed on the day the entry is made. The date\_returned column is NULL until the book is returned.

```
CREATE TABLE borrowed_book (  
  loaner_name CHAR(100) PRIMARY KEY,  
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,  
  date_returned DATE,  
  book CHAR(20),  
  FOREIGN KEY (book) REFERENCES library_books (isbn)  
)
```

The following statement creates tables for a sales database to hold order and order item information.

```
CREATE TABLE Orders (  
  order_num INTEGER NOT NULL PRIMARY KEY,  
  date_ordered DATE,  
  name CHAR(80)  
);  
CREATE TABLE Order_item (  
  order_num INTEGER NOT NULL,  
  item_num SMALLINT NOT NULL,  
  PRIMARY KEY (order_num, item_num),  
  FOREIGN KEY (order_num)  
  REFERENCES Orders (order_num)  
)
```

## Related Information

[Identifiers in UltraLite \[page 256\]](#)

[SQL Data Types \[page 288\]](#)

[GLOBAL AUTOINCREMENT Columns in UltraLite \[page 75\]](#)

[CURRENT TIMESTAMP Special Value - UltraLite \[page 261\]](#)

[CURRENT UTC TIMESTAMP Special Value - UltraLite \[page 262\]](#)

[CURRENT DATE Special Value - UltraLite \[page 260\]](#)

[CURRENT TIME Special Value - UltraLite \[page 260\]](#)

[SQL Data Types \[page 288\]](#)

[Partition Sizes \[page 77\]](#)

[UltraLite Database Schemas \[page 52\]](#)

[Reading Database Properties \[page 43\]](#)

[UltraLite global\\_database\\_id Option \[page 210\]](#)  
[NEWID Function \[Miscellaneous\] \[page 445\]](#)  
[Expressions in UltraLite \[page 264\]](#)  
[DROP TABLE Statement \[UltraLite\] \[page 548\]](#)  
[CREATE TABLE Statement](#)  
[UltraLite max\\_hash\\_size Creation Option \[page 161\]](#)

## 1.13.6.13 CREATE USER Statement [UltraLite]

Creates a database user or group.

### ☰ Syntax

```
CREATE USER user-name IDENTIFIED BY password
```

## Parameters

### **user-name**

The name of the user you are creating.

### **password**

The password for the user you are creating.

## Remarks

If you use this statement in a procedure, do not specify the password as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

- User IDs cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons
- Passwords are case-sensitive and they cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons
  - be longer than 255 bytes in length

## Side effects

None.

## Example

The following example creates a user named SQLTester with the password welcome.

```
CREATE USER SQLTester IDENTIFIED BY welcome
```

## 1.13.6.14 DELETE Statement [UltraLite]

Deletes rows from a table in the database.

### ☰ Syntax

```
DELETE [ FROM ] table-name [[AS] correlation-name]  
[ WHERE search-condition ]
```

## Parameters

### correlation-name

An identifier to use when referencing the table from elsewhere in the statement.

### WHERE clause

If a WHERE clause is specified, only rows satisfying `search-condition` are deleted.

The WHERE clause does not support non-deterministic functions (like RAND) or variables. Nor does this clause restrict columns; columns may need to reference another table when used in a subquery.

## Remarks

The way in which UltraLite traces row states is unique. Be sure you understand the implication of deletes and row states.



## Example

The following statement removes employee 105 from the Employees table.

```
DELETE
FROM Employees
WHERE EmployeeID = 105
```

The following statement removes all data before the year 2000 from the FinancialData table.

```
DELETE
FROM FinancialData
WHERE Year < 2000
```

## Related Information

[UltraLite Database Row State Management \[page 584\]](#)

[Search Conditions in UltraLite \[page 272\]](#)

[START SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 561\]](#)

[STOP SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 562\]](#)

## 1.13.6.15 DROP INDEX Statement [UltraLite]

Deletes an index.

☰ Syntax

```
DROP INDEX [ IF EXISTS ] [table-name.]index-name
```

## Remarks

You cannot drop the primary index of a table.

UltraLite does not process requests or queries referencing the index while the DROP INDEX statement is being processed. Furthermore, you cannot execute DROP INDEX when the database includes active queries or uncommitted transactions.

Use the IF EXISTS clause if you do not want an error returned when the DROP INDEX statement attempts to remove an index that does not exist.

When you specify the IF EXISTS clause and the named table cannot be located, an error is returned.

For UltraLite.NET users: You cannot execute this statement unless you also call the ULBulkCopy.Dispose method for all data objects (for example, ULDataReader).

Statements are not released if database schema changes are initiated at the same time.

## Example

The following statement deletes a fictitious index, `fin_codes_idx`, on the `FinancialData` table:

```
DROP INDEX FinancialData.fin_codes_idx
```

## Related Information

[UltraLite Indexes \[page 62\]](#)

[UltraLite Database Schemas \[page 52\]](#)

[CREATE INDEX Statement \[UltraLite\] \[page 531\]](#)

## 1.13.6.16 DROP PUBLICATION Statement [UltraLite]

Deletes publications.

☞ Syntax

```
DROP PUBLICATION [ IF EXISTS ] publication-name, ...
```

## Remarks

Use the `IF EXISTS` clause if you do not want an error returned when the `DROP PUBLICATION` statement attempts to remove a publication that does not exist.

## Example

The following statement drops the `pub_contact` publication.

```
DROP PUBLICATION pub_contact
```

## Related Information

[UltraLite Client Synchronization Design \[page 79\]](#)

[ALTER PUBLICATION Statement \[UltraLite\] \[page 521\]](#)

[CREATE PUBLICATION Statement \[UltraLite\] \[page 533\]](#)

### 1.13.6.17 DROP SYNCHRONIZATION PROFILE Statement [UltraLite]

Deletes a synchronization profile.

☞ Syntax

```
DROP SYNCHRONIZATION PROFILE [ IF EXISTS ] sync-profile-name
```

#### Parameters

**sync-profile-name**

The name of the synchronization profile.

#### Remarks

Synchronization profiles define how an UltraLite database synchronizes with the MobiLink server.

Use the IF EXISTS clause if you do not want an error returned when the DROP SYNCHRONIZATION PROFILE statement attempts to remove a synchronization profile that does not exist.

#### Side effects

None.

#### Example

The following example shows the syntax for dropping a synchronization profile called Test1.

```
DROP SYNCHRONIZATION PROFILE Test1
```

## Related Information

[CREATE SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 535\]](#)

[ALTER SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 522\]](#)

[SYNCHRONIZE Statement \[UltraLite\] \[page 563\]](#)

### 1.13.6.18 DROP TABLE Statement [UltraLite]

Removes a table, and all its data, from a database.

#### ☰, Syntax

```
DROP TABLE [ IF EXISTS ] table-name
```

## Remarks

The DROP TABLE statement drops the specified table from the database. All data in the table and any indexes and keys are also removed.

UltraLite does not process requests or queries referencing the table, or its indexes, while the DROP TABLE statement is being processed. Furthermore, you cannot execute DROP TABLE when there are active queries or uncommitted transactions.

Use the IF EXISTS clause if you do not want an error returned when the DROP TABLE statement attempts to remove a table that does not exist.

For UltraLite.NET, you cannot execute this statement unless you also call the ULBulkCopy.Dispose method for all data objects (for example, ULDataReader).

Statements are not released if schema changes are initiated at the same time.

## Example

The following statement deletes a fictitious table, EmployeeBenefits, from the database:

```
DROP TABLE EmployeeBenefits
```

## Related Information

[UltraLite Database Schemas \[page 52\]](#)

[ALTER TABLE Statement \[UltraLite\] \[page 524\]](#)

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

## 1.13.6.19 DROP USER Statement [UltraLite]

Drops a user.

☰ Syntax

```
DROP USER userid IDENTIFIED BY password
```

### Parameters

**userid**

The user ID of the user you are dropping.

**password**

The password for the user.

### Remarks

If you use this statement in a procedure, do not specify the password as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

### Side effects

None.

### Example

The following example drops the user SQLTester from a database.

```
DROP USER SQLTester
```

## Related Information

[ALTER USER Statement \[UltraLite\] \[page 528\]](#)

[CREATE USER Statement \[UltraLite\] \[page 543\]](#)

### 1.13.6.20 FROM Clause [UltraLite]

Use this clause to specify the tables or views involved in a SELECT statement.

#### ≡ Syntax

```
FROM table-expression, ...
```

```
table-expression :  
table-name [ [ AS ] correlation-name ]  
| ( select-list ) [ AS ] derived-table-name ( column-name, ... )  
| ( table-expression )  
| table-expression join-operator table-expression [ ON search-condition ] ...
```

```
join-operator :
```

```
'  
| INNER JOIN  
| CROSS JOIN  
| LEFT OUTER JOIN  
| JOIN
```

## Parameters

### table-name

A base table or temporary table. Tables cannot be owned by different users in UltraLite. If you qualify tables with user ID, the ID is ignored.

### correlation-name

An identifier to use when referencing the table from elsewhere in the statement. For example, in the following statement, a is defined as the correlation name for the Contacts table, and b is the correlation name for the Customers table.

```
SELECT *  
FROM Contacts a, Customers b  
WHERE a.CustomerID=b.ID
```

### derived-table-name

A derived table is a nested SELECT statement in the FROM clause.

Items from the SELECT list of the derived table are referenced by the (optional) derived table name followed by a period (.) and the column name. You can use the column name by itself if it is unambiguous.

You cannot reference derived tables from within the SELECT statement.

### join-operator

Specify the type of join. If you specify a comma (,), or CROSS JOIN, you cannot specify an ON subclause. If you specify JOIN, you must specify an ON subclause. For INNER JOIN and LEFT OUTER JOIN, the ON clause is optional.

## Remarks

When there is no FROM clause, the expressions in the SELECT statement must be a constant expression.

### i Note

Although this description refers to tables, it also applies to derived tables unless otherwise noted.

The FROM clause creates a result set consisting of all the columns from all the tables specified. Initially, all combinations of rows in the specified tables are in the result set, and the number of combinations is usually reduced by JOIN conditions and/or WHERE conditions.

If you do not specify the type of join, and instead list the tables as a comma-separated list, a CROSS JOIN is used, by default.

For INNER joins, restricting results of the join using an ON clause or WHERE clause returns equivalent results. For OUTER joins, they are not equivalent.

### i Note

UltraLite does not support KEY JOINS nor NATURAL joins.

## Example

The following are valid FROM clauses:

```
...  
FROM Employees  
...
```

```
...  
FROM Customers  
CROSS JOIN SalesOrders  
CROSS JOIN SalesOrderItems  
CROSS JOIN Products  
...
```

The following query uses a derived table to return the names of the customers in the Customers table who have more than three orders in the SalesOrders table:

```
SELECT Surname, GivenName, number_of_orders  
FROM Customers JOIN  
    ( SELECT CustomerID, COUNT(*)  
      FROM SalesOrders
```

```
GROUP BY CustomerID )
AS sales_order_counts( CustomerID, number_of_orders )
ON ( Customers.id = sales_order_counts.CustomerID )
WHERE number_of_orders > 3
```

## Related Information

[Joins: Retrieving Data from Several Tables](#)

[Subqueries in Expressions - UltraLite \[page 270\]](#)

[DELETE Statement \[UltraLite\] \[page 544\]](#)

[SELECT Statement \[UltraLite\] \[page 558\]](#)

[UPDATE Statement \[UltraLite\] \[page 567\]](#)

### 1.13.6.21 INSERT Statement [UltraLite]

Inserts rows into a table.

#### Syntax

```
INSERT [ INTO ]
table-name [ ( column-name, ... ) ]
{ VALUES( expression, ... ) | select-statement }
```

## Remarks

The INSERT statement can be used to insert a single row, or to insert multiple rows from a query result set.

If columns are specified, values are inserted one for one into the specified columns. If the list of column names is not specified, values are inserted into the table columns in the order in which they appear in the table (the same order as retrieved with SELECT \*). Rows are inserted into the table in an arbitrary order.

Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive.

## Example

The following statement adds an Eastern Sales department to the database.

```
INSERT
INTO Departments ( DepartmentID, DepartmentName )
VALUES ( 230, 'Eastern Sales' )
```



The following statement adds the values of a and b into mytable from othertable where the value of c in othertable is greater than 10.

```
INSERT INTO mytable( col1, col2 ) SELECT a, b FROM othertable WHERE c > 10
```

## Related Information

[SELECT Statement \[UltraLite\] \[page 558\]](#)

### 1.13.6.22 LOAD TABLE Statement [UltraLite]

Imports bulk data into a database table from an external file.

#### Syntax

```
LOAD [ INTO ] TABLE [ owner.]table-name  
( column-name, ... )  
FROM stringfilename  
[ load-option ... ]
```

```
load-option :  
CHECK CONSTRAINTS { ON | OFF }  
| COMPUTES { ON | OFF }  
| DEFAULTS { ON | OFF }  
| DELIMITED BY string  
| ENCODING encoding  
| ESCAPES { ON }  
| FORMAT { ASCII | TEXT }  
| ORDER { ON | OFF }  
| QUOTES { ON | OFF }  
| SKIP integer  
| STRIP { ON | OFF | BOTH }  
| WITH CHECKPOINT { ON | OFF }
```

```
comment-prefix : string
```

```
encoding : string
```

## Parameters

### column-name

Use this clause to specify one or more columns to load data into. Any columns not present in the column list become NULL if DEFAULTS is OFF. If DEFAULTS is ON and the column has a default value, that value is used. If DEFAULTS is OFF and a non-nullable column is omitted from the column list, the database server attempts to convert the empty string to the column's type.

When a column list is specified, it lists the columns that are expected to exist in the file and the order in which they are expected to appear. Column names cannot be repeated.

### **FROM string-filename**

Use this to specify a file from which to load the data. The `string-filename` is passed to the database server as a string. The string is therefore subject to the same database formatting requirements as other SQL strings. In particular:

- To indicate directory paths, the backslash character (\) must be represented by two backslashes. For example, the `string-filename` that loads data from the `c:\temp\input.dat` file into the Employees table is `c:\\temp\\input.dat`.
- The path name is relative to the database server, not to the client application.
- You can use UNC path names to load data from files on computers other than the database server.

### **load-option clause**

There are several load options you can specify to control how data is loaded. The following list gives the supported load options:

#### **CHECK CONSTRAINTS clause**

This clause controls whether constraints are checked during loading. CHECK CONSTRAINTS is ON by default, but the Unload utility (ulunload) writes out LOAD TABLE statements with CHECK CONSTRAINTS set to OFF. Setting CHECK CONSTRAINTS to OFF disables check constraints, which can be useful, for example, during database rebuilding.

#### **COMPUTES clause**

This option is processed but ignored by UltraLite.

#### **DEFAULTS clause**

By default, DEFAULTS is set to OFF. If DEFAULTS is OFF, any column not present in the list of columns is assigned NULL. If DEFAULTS is set to OFF and a non-nullable column is omitted from the list, the database server attempts to convert the empty string to the column's type. If DEFAULTS is set to ON and the column has a default value, that value is used.

#### **DELIMITED BY clause**

Use this clause to specify the column delimiter string. The default column delimiter string is a comma; however, it can be any string up to 255 bytes in length (for example, `... DELIMITED BY '###' ...`). The formatting requirements of other SQL strings apply. To specify tab-delimited values, you could specify the hexadecimal escape sequence for the tab character (9), `... DELIMITED BY '\x09' ...`.

#### **ENCODING clause**

This clause specifies the character encoding used for the data being loaded into the database. UltraLite does not perform character set translation: the encoding of the data file must match the database.

#### **ESCAPES clause**

ESCAPES is always ON, therefore characters following the backslash character are recognized and interpreted as special characters by the database server. Newline characters can be included as the combination `\n`, and other characters can be included in data as hexadecimal ASCII codes, such as `\x09` for the tab character. A sequence of two backslash characters (`\\`) is interpreted as a single backslash. A backslash followed by any character other than `n`, `x`, `X`, or `\` is interpreted as two separate characters. For example, `\q` inserts a backslash and the letter `q`.

### FORMAT clause

This clause specifies the format of the data source you are loading data from. With TEXT, input lines are assumed to be characters (as defined by the ENCODING option), one row per line, with values separated by the column delimiter string. ASCII is also supported.

### QUOTES clause

This clause specifies whether strings are enclosed in quotes. UltraLite only supports ON, therefore the LOAD TABLE statement expects strings to be enclosed in quote characters. The quote character is an apostrophe (single quote). The first such character encountered in a string is treated as the quote character for the string. Strings must be terminated by a matching quote.

Column delimiter strings can be included in column values. Also, quote characters are assumed not to be part of the value. Therefore, the following line is treated as two values, not three, despite the presence of the comma in the address. Also, the quotes surrounding the address are not inserted into the database.

```
'123 High Street, Anytown', (715) 398-2354
```

To include a quote character in a value, you must use two quotes. The following line includes a value in the third column that is a single quote character:

```
'123 High Street, Anytown', '(715) 398-2354', ''''
```

### SKIP clause

Use this clause to specify whether to ignore lines at the beginning of a file. The `integer` argument specifies the number of lines to skip. You can use this clause to skip over a line containing column headings, for example.

### STRIP clause

This clause is processed but ignored. This clause specifies whether unquoted values should have leading or trailing blanks stripped off before they are inserted. The STRIP option accepts the following options:

#### STRIP ON

Strip leading blanks.

#### STRIP OFF

Do not strip off leading or trailing blanks.

#### STRIP BOTH

Strip both leading and trailing blanks.

### WITH CHECKPOINT clause

Use this clause to specify whether to perform a checkpoint. The default setting is OFF. If this clause is set to ON, a checkpoint is issued after successfully completing the statement.

## Remarks

This statement also provides support for handling the output of the SQL Anywhere dbunload utility (the reload.sql file). LOAD TABLE is only available using DBISQL on Windows.

The recommended method for unloading and reloading UltraLite databases is to use the `ulunload` and `ulload` utilities. Note also that the `ulunit` utility can load schema and data directly from a SQL Anywhere database.

`LOAD TABLE` allows efficient mass insertion into a database table from a file. It is provided primarily as a means of supporting the output of the SQL Anywhere `dbunload` utility (the `reload.sql` file).

With `FORMAT TEXT`, a `NULL` value is indicated by specifying no value. For example, if three values are expected and the file contains `1, , 'Fred', ,`, then the values inserted are 1, `NULL`, and Fred. If the file contains `1, 2, ,`, then the values 1, 2, and `NULL` are inserted. Values that consist only of spaces are also considered `NULL` values. For example, if the file contains `1, , 'Fred', ,`, then values 1, `NULL`, and Fred are inserted. All other values are considered not `NULL`. For example, `"` (a single quote followed by single quote) is an empty string. `'NULL'` is a string containing four letters.

If a column being loaded by `LOAD TABLE` does not allow `NULL` values and the file value is `NULL`, then numeric columns are given the value 0 (zero), character columns are given an empty string (`"`). If a column being loaded by `LOAD TABLE` allows `NULL` values and the file value is `NULL`, then the column value is `NULL` (for all types).

If the table contains columns `a`, `b`, and `c`, and the input data contains `a`, `b`, and `c`, but the `LOAD` statement only specifies `a` and `b` as columns to load data into, the following values are inserted into column `c`:

- if `DEFAULTS ON` is specified, and column `c` has a default value, the default value is used.
- if column `c` does not have a default value, and `NULLs` are allowed, a `NULL` is used.
- if column `c` has no default value and does not allow `NULLs`, either a zero (0) or an empty string (`"`), is used, or an error is returned, depending on the data type of the column.

## Side effects

Automatic commit.

## Example

Following is an example of `LOAD TABLE`. First, you create a table, and then load data into it using a file called `input.txt`.

```
CREATE TABLE t( a CHAR(100) primary key, let_me_default INT DEFAULT 1, c
CHAR(100) )
```

Following is the content of a file called `input.txt`:

```
'this_is_for_column_c', 'this_is_for_column_a', ignore_me
```

The following `LOAD` statement loads the file called `input.txt`:

```
LOAD TABLE T ( c, a ) FROM 'input.txt' FORMAT TEXT DEFAULTS ON
```

The command `SELECT * FROM t` yields the result set:

a	let_me_default	c
this_is_for_column_a	1	this_is_for_column_c

## Related Information

[INSERT Statement \[UltraLite\] \[page 552\]](#)

[UltraLite Database Unload Utility \(ulunload\) \[page 243\]](#)

[Unload Utility \(dbunload\)](#)

## 1.13.6.23 ROLLBACK Statement [UltraLite]

Ends a transaction and reverts any changes made to data since the last COMMIT or ROLLBACK statement was executed.

☰ Syntax

```
ROLLBACK [ WORK ]
```

## Remarks

Using UltraLite SQL creates a transaction. A transaction consists of all changes (INSERTs, UPDATEs, and DELETEs) since the last ROLLBACK or COMMIT. The ROLLBACK statement ends the current transaction and undoes all changes made to the database since the previous COMMIT or ROLLBACK.

## Example

The following statement rolls the database back to the state it was in at the previous commit:

```
ROLLBACK
```

## Related Information

[COMMIT Statement \[UltraLite\] \[page 530\]](#)

## 1.13.6.24 SELECT Statement [UltraLite]

Retrieves information from the database.

### ☰ Syntax

```
SELECT [ DISTINCT ] [ row-limitation ]
select-list
[ FROM table-expression, ... ]
[ WHERE search-condition ]
[ GROUP BY group-by-expression, ... ]
[ ORDER BY order-by-expression, ... ]
[ FOR { UPDATE | READ ONLY } ]
[ OPTION ( FORCE ORDER ) ]
```

```
row-limitation :
FIRST
| TOP n [ START AT m ]
```

```
select-list :
expression [ [ AS ] alias-name ], ...
```

```
order-by-expression :
{ integer | expression } [ ASC | DESC ]
```

## Parameters

### **DISTINCT clause**

Specify DISTINCT to eliminate duplicate rows from the results. If you do not specify DISTINCT, all rows that satisfy the clauses of the SELECT statement are returned, including duplicate rows. Many statements take significantly longer to execute when DISTINCT is specified, so you should reserve DISTINCT for cases where it is necessary.

### **row-limitation clause**

Use row limitations to return a subset of the results. For example, specify FIRST to retrieve the first row of a result set. Use TOP<sub>n</sub> to return the first *n* rows of the results. Specify START AT<sub>m</sub> to control the location of the starting row when retrieving the TOP<sub>n</sub> rows. To order the rows so that these clauses return meaningful results, specify an ORDER BY clause for the SELECT statement.

### **select-list**

A list of expressions specifying what to retrieve from the database. Usually, the expressions in a SELECT list are column names. However, they can be other types of expressions, such as functions. Use an asterisk (\*) to select all columns of all tables listed in the FROM clause. Optionally, you can define an alias for each expression in the *select-list*. Using an alias allows you to reference the *select-list* expressions from elsewhere in the query, such as from within the WHERE and ORDER BY clauses.

### **FROM clause**

Rows are retrieved from the tables and views specified in the *table-expression*.

### **WHERE clause**

If a WHERE clause is specified, only rows satisfying `search-condition` are selected.

#### **GROUP BY clause**

The result of the query that has a GROUP BY clause contains one row for each distinct set of values in the GROUP BY expression. The resulting rows are often referred to as groups since there is one row in the result for each group of rows from the table list. Aggregate functions can be applied to the rows in these groups. NULL is considered to be a unique value if it occurs.

#### **ORDER BY clause**

This clause sorts the results of a query according to the expression specified in the clause. Each expression in the ORDER BY clause can be sorted in ascending (ASC) or descending (DESC) order (the default). If the expression is an integer `n`, then the query results are sorted by the `n`th expression in the SELECT list.

The only way to ensure that rows are returned in a particular order is to use ORDER BY. In the absence of an ORDER BY clause, UltraLite returns rows in whatever order is most efficient.

UltraLite does not support the ordering of LONG VARCHAR or LONG BINARY values.

#### **FOR clause**

This clause has two variations that control the query's behavior:

##### **FOR READ ONLY**

This clause indicates the query is not being used for updates. Specify this clause whenever possible because UltraLite can sometimes achieve better performance when it is known that a query is not going to be used for updates. For example, UltraLite could perform a direct table scan when it learns that read-only access is required. FOR READ ONLY is the default behavior.

##### **FOR UPDATE**

This clause allows the query to be used for updates. This clause must be explicitly specified otherwise updates are not permitted (FOR READ ONLY is the default behavior).

#### **OPTION ( FORCE ORDER ) clause**

This clause is not recommended for general use. It overrides the UltraLite choice of the order in which to access tables, and requires that UltraLite access the tables in the order they appear in the query. Only use this clause when the query order is determined to be more efficient than the UltraLite order.

UltraLite can also use execution plans to optimize queries.

## **Remarks**

Always remember to close the query. Otherwise memory cannot be freed and the number of temporary tables that remain can proliferate unnecessarily.

## **Example**

The following statement selects the number of employees from the Employees table.

```
SELECT COUNT (*)  
FROM Employees
```

The following statement selects 10 rows from the Employees table starting from the 40<sup>th</sup> row and ending at the 49<sup>th</sup> row.

```
SELECT TOP 10 START AT 40 * FROM Employees
```

## Related Information

[UltraLite Performance Tips \[page 569\]](#)

[Queries](#)

[Execution Plans in UltraLite \[page 575\]](#)

[Direct Page Scans \[page 578\]](#)

[SELECT Statement](#)

[Search Conditions in UltraLite \[page 272\]](#)

[FROM Clause \[UltraLite\] \[page 550\]](#)

### 1.13.6.25 SET OPTION Statement [UltraLite]

Changes the values of database options.

#### ≡ Syntax

```
SET OPTION option-name=[option-value]
```

```
option-name: identifier
```

```
option-value: string, identifier, or number
```

## Remarks

This statement allows you to set options on an UltraLite database. Most UltraLite options are set when the database is initially created and cannot be modified afterward.

You cannot specify whether an option is persistent or not. UltraLite determines whether it is a persistent or temporary option. Persistent options are stored in the database. Temporary options are used only until the connection or database stops running.

UltraLite performs a commit when persistent options are set: `global_database_id` and `ml_remote_id`. UltraLite does not perform a commit on temporary or connection-based options.

The only database option that can be unset is `ml_remote_id`. For example:

```
SET OPTION ml_remote_id=
```



The result is that the ID is set to NULL. When this occurs, UltraLite will automatically generate a new value at the next synchronization.

## Example

The following statement sets the `global_database_id` option to 100:

```
SET OPTION global_database_id=100
```

## Related Information

[UltraLite Database Options \[page 206\]](#)

[UltraLite `global\_database\_id` Option \[page 210\]](#)

[UltraLite `ml\_remote\_id` Option \[page 211\]](#)

[DB\\_PROPERTY Function \[System\] \[page 390\]](#)

## 1.13.6.26 START SYNCHRONIZATION DELETE Statement [UltraLite]

Restarts the logging of deleted rows for MobiLink synchronization.

⇒ Syntax

```
START SYNCHRONIZATION DELETE
```

## Remarks

UltraLite databases automatically log changes made to rows that need to be synchronized. UltraLite uploads these changes to the consolidated database during the next synchronization. This statement allows you to restart logging of deleted rows, previously stopped by a `STOP SYNCHRONIZATION DELETE` statement.

When a `STOP SYNCHRONIZATION DELETE` statement is executed, none of the delete operations executed on that connection are synchronized. The effect continues until a `START SYNCHRONIZATION DELETE` statement is executed.

Do not use `START SYNCHRONIZATION DELETE` if your application does not synchronize data.

The way in which UltraLite traces row states is unique. Be sure you understand the implication of deletes and row states.

## Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION DELETE and STOP SYNCHRONIZATION DELETE.

```
STOP SYNCHRONIZATION DELETE;
DELETE FROM PROPOSAL
  WHERE last_modified < months( CURRENT_TIMESTAMP, -1 );
START SYNCHRONIZATION DELETE;
COMMIT;
```


## Related Information

[UltraLite Database Row State Management \[page 584\]](#)

[STOP SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 562\]](#)

## 1.13.6.27 STOP SYNCHRONIZATION DELETE Statement [UltraLite]

Stops the logging of deleted rows for MobiLink synchronization.

 Syntax

*STOP SYNCHRONIZATION DELETE*

## Remarks

UltraLite databases automatically log changes made to rows that need to be synchronized. UltraLite uploads these changes to the consolidated database during the next synchronization. This statement allows you to stop the logging of deleted rows, previously started using a STOP SYNCHRONIZATION DELETE statement. This command can be useful when making corrections to a remote database, but should be used with caution as it effectively disables MobiLink synchronization. You should only stop deletion logging temporarily.

When a STOP SYNCHRONIZATION DELETE statement is executed, no further delete operations executed on that connection are synchronized. The effect continues until a START SYNCHRONIZATION DELETE statement is executed.

Do not use STOP SYNCHRONIZATION DELETE if your application does not synchronize data.

The way in which UltraLite traces row states is unique. Be sure you understand the implication of deletes and row states.

## Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION DELETE and STOP SYNCHRONIZATION DELETE.

```
STOP SYNCHRONIZATION DELETE;
DELETE FROM PROPOSAL
WHERE last_modified < months( CURRENT_TIMESTAMP, -1 );
START SYNCHRONIZATION DELETE;
COMMIT;
```

## Related Information

[UltraLite Database Row State Management \[page 584\]](#)

[START SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 561\]](#)

## 1.13.6.28 SYNCHRONIZE Statement [UltraLite]

Synchronize an UltraLite database via the MobiLink server.

### ⌘ Syntax

```
SYNCHRONIZE {
  PROFILE sync-profile-name [ MERGE sync-option [ ;... ] ]
  | USING sync-option [ ;... ]
}
```

```
sync-option :
sync-option-name = sync-option-value
```

```
sync-option-name : string
```

```
sync-option-value : string
```

## Parameters

### **sync-profile-name**

The name of the synchronization profile.

### **MERGE clause**

Use this clause when you want to add or override options that are provided in the synchronization profile.

### **USING clause**

Use this clause when you want to specify the synchronization options without referencing a synchronization profile.

**sync-option**

A string of one or more synchronization option value pairs, separated by semicolons. For example, 'option1=value1;option2=value2'.

**sync-option-name**

The name of the synchronization option.

**sync-option-value**

The value for the synchronization option.

## Remarks

The synchronization is configured according to the parameters in the synchronization profile, or the parameters can be specified in the statement itself.

By allowing synchronization options to be merged in, developers can choose to omit storing some options in the database (like the MobiLinkPwd for instance).

If a synchronization callback function is defined and registered with UltraLite, whenever a SYNCHRONIZE statement is executed, progress information for that synchronization is passed to the callback function. If no callback is registered, progress information is suppressed.

## Side effects

None.

## Example

The following example shows the syntax for synchronizing a synchronization profile called Test1 where the MobiLinkPwd has not been stored as part of the profile:

```
SYNCHRONIZE PROFILE Test1 MERGE 'MobiLinkPwd=sql'
```

The following example shows the syntax for adding the publication and uploadonly options to a synchronization profile called Test1.

```
SYNCHRONIZE PROFILE Test1  
MERGE 'publications=p2;uploadonly=on'
```

The following example illustrates how to use USING.

```
SYNCHRONIZE USING  
'MobiLinkUid=joe;MobiLinkPwd=sql;ScriptVersion=1;Stream=TCPIP{host=localhost}'
```

The following example shows the syntax for synchronizing the publication and uploadonly options.

```
SYNCHRONIZE
  USING 'publications=p2;uploadonly=on'
```

## Related Information

[CREATE SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 535\]](#)

[ALTER SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 522\]](#)

[DROP SYNCHRONIZATION PROFILE Statement \[UltraLite\] \[page 547\]](#)

## 1.13.6.29 TRUNCATE TABLE Statement [UltraLite]

Deletes all rows from a table without deleting the table.

### Syntax

```
TRUNCATE TABLE table-name
```

## Remarks

The TRUNCATE TABLE statement deletes all rows from a table and the MobiLink server is not informed of their removal upon subsequent synchronization. It is equivalent to executing the following statements:

```
STOP SYNCHRONIZATION DELETE;
DELETE FROM TABLE;
START SYNCHRONIZATION DELETE;
```

### Note

This statement should be used with great care on a database involved in synchronization or replication. Because the MobiLink server is not notified, this deletion can lead to inconsistencies that can cause synchronization or replication to fail.

After a TRUNCATE TABLE statement, the table structure, all the indexes, and the constraints and column definitions continue to exist; only data is deleted.

TRUNCATE TABLE cannot execute if a statement that affects the table is already being referenced by another request or query. Similarly, UltraLite does not process requests referencing the table while that table is being altered. Furthermore, you cannot execute TRUNCATE TABLE when the database includes active queries or uncommitted transactions.

For UltraLite.NET users: You cannot execute this statement unless you also call the `ULBulkCopy.Dispose` method for all data objects (for example, `ULDataReader`).

## Schema changes

Statements are not released if schema changes are initiated at the same time.

## Side effects

If the table contains a column defined as `DEFAULT AUTOINCREMENT` or `DEFAULT GLOBAL AUTOINCREMENT`, `TRUNCATE TABLE` resets the next available value for the column.

Once rows are marked as deleted with `TRUNCATE TABLE`, they are no longer accessible to the user who performed this action, unless the user issues a `ROLLBACK` statement. However, they do remain accessible from other connections. Use `COMMIT` to make the deletion permanent, thereby making the data inaccessible from all connections.

If you synchronize the truncated table, all `INSERT` statements applied to the table take precedence over a `TRUNCATE TABLE` statement.

## Example

The following statement deletes all rows from the `Departments` table.

```
TRUNCATE TABLE Departments
```

If you execute this example, be sure to execute a `ROLLBACK` statement to revert your change.

## Related Information

[UltraLite Database Schemas \[page 52\]](#)

[DELETE Statement \[UltraLite\] \[page 544\]](#)

[START SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 561\]](#)

[STOP SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 562\]](#)

## 1.13.6.30 UNION Statement [UltraLite]

Combines the results of two or more select statements.

### ☞ Syntax

```
select-statement-without-ordering  
[ UNION [ ALL | DISTINCT ] select-statement-without-ordering ]...  
[ ORDER BY [ number [ ASC | DESC ] , ... ]
```

## Remarks

The results of several SELECT statements can be combined into a larger result using UNION. Each SELECT statement must have the same number of expressions in their respective SELECT list, and cannot contain an ORDER BY clause.

The results of UNION ALL are the combined results of the unioned SELECT statements. Specify UNION or UNION DISTINCT to get results without duplicate rows; however, removing duplicate rows adds to the total execution time for the statement. Specify UNION ALL to allow duplicate rows.

When attempting to combine corresponding expressions that are of different data types, UltraLite attempts to find a data type in which to represent the combined values. If this is not possible, the union operation fails and an error is returned (for example "Cannot convert 'Surname' to a numeric").

The column names displayed in the results are column names (or aliases) used for the first SELECT statement.

ORDER BY for UNION is restricted to the integer format. The ORDER BY clause uses integers to establish the ordering, where the integer indicates the query expression(s) on which to sort the results.

## Example

The following example lists all distinct surnames found in the Employees and Customers tables, combined.

```
SELECT Surname FROM Employees
UNION
SELECT Surname FROM Customers
```

## Related Information

[SELECT Statement \[UltraLite\] \[page 558\]](#)

### 1.13.6.31 UPDATE Statement [UltraLite]

Modifies rows in a table.

#### ☰ Syntax

```
UPDATE table-name [[AS] correlation-name]
SET column-name = expression, ...
[ WHERE search-condition ]
```

## Parameters

### **table-name**

The `table-name` specifies the name of the table to update. Only a single table is allowed.

### **correlation-name**

An identifier to use when referencing the table from elsewhere in the statement.

### **SET clause**

Each named column is set to the value of the expression on the right side of the equal sign. There are no restrictions on the expression. If the expression is a `column-name`, the old value is used.

Only columns specified in the SET clause have their values changed. In particular, you cannot use UPDATE to set a column's value to its default.

### **WHERE clause**

If a WHERE clause is specified, only rows satisfying `search-condition` are updated.

## Remarks

The UPDATE statement modifies values in a table.

Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive.

## Example

The following statement transfers employee Philip Chin (employee 129) from the sales department to the marketing department (department 400).

```
UPDATE Employees
SET DepartmentID = 400
WHERE EmployeeID = 129
```

An example using `correlation-name`.

```
UPDATE Employee E
SET salary = salary * 1.05
WHERE EXISTS( SELECT 1 FROM Sales S HAVING E.Sales > Avg( S.sales)
GROUP BY S.dept_no )
```

## Related Information

[INSERT Statement \[UltraLite\] \[page 552\]](#)

[DELETE Statement \[UltraLite\] \[page 544\]](#)



## 1.14 UltraLite Performance Tips

Several topics are provided to help improve performance of UltraLite databases.

### In this section:

#### [Cache Size Adjustment for an UltraLite Database \[page 569\]](#)

Although adjusting the cache size is not required, you should adjust the cache size when your UltraLite database application is requested to reduce its memory usage by the operating system on mobile devices.

#### [Query Performance Tips \[page 570\]](#)

Several topics are provided to help improve performance of queries in UltraLite databases.

#### [Insert and Update Performance Tips \[page 582\]](#)

Several topics are provided to help improve performance of inserts and updates in UltraLite databases.

#### [UltraLite Benchmark Tips \[page 587\]](#)

Benchmark testing activity is generally performed before reaching the production stage of the application development cycle.

### 1.14.1 Cache Size Adjustment for an UltraLite Database

Although adjusting the cache size is not required, you should adjust the cache size when your UltraLite database application is requested to reduce its memory usage by the operating system on mobile devices.

UltraLite database cache sizes increase dynamically in response to data operations and as available device memory allows within the parameters you specify. Normally you do not need to specify any parameters. If your database is large (400 MB for instance), you may want to specify the `CACHE_MAX_SIZE` parameter to raise the maximum limit beyond the default. UltraLite allocates some data structures based on the maximum cache size, so the default is not extremely large: you must explicitly request a large maximum to incur this extra memory overhead. There is no benefit to specifying a maximum cache size that is much larger than your maximum actual database file size.

UltraLite does not shrink the cache automatically. The database cache size can only be controlled explicitly in your application with the `cache_allocation` database option. In response to a low memory event raised by the operating system, adjust the `cache_allocation` database option after connecting to the database.

### Example

The following UltraLite C++ code sample illustrates how to set the maximum cache size to 100 MB by updating the connection string:

```
static ul_char const * ConnectionParms =
```

```
"UID=DBA;PWD=sql;DBF=sample.udb;CACHE_MAX_SIZE=100m";
```

The following UltraLite C++ code sample illustrates how to reduce the cache allocation in half to resize the cache:

```
ULConnection * conn = ULDatabaseManager::OpenConnection(ConnectionParms);  
ul_u_long percent;  
percent = conn->GetDatabasePropertyInt( "cache_allocation" );  
conn->SetDatabaseOptionInt( "cache_allocation", percent / 2 );
```

## Related Information

[UltraLite cache\\_allocation Option \[page 207\]](#)

[UltraLite CACHE\\_SIZE Connection Parameter \[page 187\]](#)

[UltraLite CACHE\\_MIN\\_SIZE Connection Parameter \[page 186\]](#)

[UltraLite CACHE\\_MAX\\_SIZE Connection Parameter \[page 185\]](#)

## 1.14.2 Query Performance Tips

Several topics are provided to help improve performance of queries in UltraLite databases.

**In this section:**

[Index Scan Creation and Maintenance \[page 570\]](#)

You can create one or more indexes to improve the performance of your queries, or to ensure that row values remain unique.

[Index Hashing \[page 571\]](#)

You can tune the performance of your queries by choosing a specific size for the maximum **hash**.

[Optimal Hash Size Limit \[page 573\]](#)

The UltraLite default maximum hash size of 4 bytes suits most deployments.

[Execution Plans in UltraLite \[page 575\]](#)

UltraLite execution plans show how tables and indexes are accessed when a query is executed.

### 1.14.2.1 Index Scan Creation and Maintenance

You can create one or more indexes to improve the performance of your queries, or to ensure that row values remain unique.

An index provides an ordering of a table's rows based on the values in some or all of the columns. When creating indexes, the order in which you select columns to be indexed becomes the order in which the columns actually appear in the index. Indexes can greatly improve the performance of searches on the indexed column(s) when used strategically.

Use the following recommended practices for improving query performance:

- Create an index on any column:
  - for values that you search for on a regular basis
  - that the query uses to join tables
  - that are commonly used in ORDER BY, GROUP BY, or WHERE clauses
- Create a composite index and ensure that the first column of the index is used most often by the predicate in your query when creating it.
- Ensure that the update maintenance overhead an index introduces is not too high for the memory of your device.
- Do not create or maintain unnecessary indexes. Indexes must be updated when the data in a column is modified, so all insert, update, and delete operations are performed on the indexes as well.
- Create an index on large tables.
- Do not create redundant indexes. For example, if you create an index on table T with columns (x, y), you can create a redundancy if there is another existing index on T with columns (x, y, z).

## Related Information

[UltraLite Indexes \[page 62\]](#)

[Manage Temporary Tables \[page 577\]](#)

[Direct Page Scans \[page 578\]](#)

[Viewing an Execution Plan \[page 579\]](#)

[EXPLANATION Function \[Miscellaneous\] \[page 396\]](#)

### 1.14.2.2 Index Hashing

You can tune the performance of your queries by choosing a specific size for the maximum **hash**.

A hash key represents the actual values of the indexed column. An index hash key aims to avoid the expensive operation of finding, loading, and then unpacking the rows to determine the indexed value. It prevents these operations by including enough of the actual row data with a row ID.

A row ID allows UltraLite to locate the actual row data in the database file. If you set the hash size to 0 (which disables index hashing), then the index entry only contains this row ID. If you set the hash size to anything other than 0, then a hash key is also used. A hash key can contain all or part of the transformed data in that row, and is stored with the row ID in the index page.

How much row data the hash key includes is determined by:

- The maximum hash size property you configure.
- How much is actually needed for the data type of the column.

## A Hash Example

The value of an index hash maintains the order of the actual row data of indexed columns. For example, if you have indexed a LastName column for a table called Employees, you may see four names ordered as follows:

- Anders
- Anderseck
- Andersen
- Anderson

If you hashed the first six letters, your hash keys for these row values would appear as follows:

- Anders
- Anders
- Anders
- Anders

While these entries look the same, the first Anders in the list is used to represent the actual row value of **Anders**. The last Anders in the list, however, is used to represent the actual row value **Anderson**.

Consider the following statement:

```
SELECT *
FROM Employees
WHERE LastName = 'Andersen'
```

If the Employees table only contained a very high proportion of names similar to Andersen, then the hash key may not offer enough uniqueness to gain any performance benefits. In this case, UltraLite cannot determine if any of the hash keys actually meets the conditions of this statement. When duplicate index hash keys exist, UltraLite still needs to:

1. Find the table row that matches the row ID in question.
2. Load and then unpack the data so the value can be evaluated.

Performance benefits only occur when UltraLite can discern a proportionate number of unique hash so that the query condition evaluation is immediate to the index itself. For example, if the Employees table had thousands of names, there is still enough benefit to be gained by a hash of six letters. However, if the Employees table only contained an inordinate number of names that begin with Anders\*, then you should hash at least seven letters so the degree of unique keys increases. Therefore, the original four names at the start of this example now are now represented with these hash keys:

- Anders
- Anderse
- Anderse
- Anderso

In this example, only two of the four row values would need to be unpacked and evaluated, rather than all four.

## Related Information

[Optimal Hash Size Limit \[page 573\]](#)

[Adding an UltraLite Index \[page 65\]](#)

[UltraLite max\\_hash\\_size Creation Option \[page 161\]](#)

### 1.14.2.3 Optimal Hash Size Limit

The UltraLite default maximum hash size of 4 bytes suits most deployments.

You can increase the size to include more data with the row ID. However, this change could increase the size of the index and fragment it among multiple pages. This change can possibly increase the size of the database as a result. The impact of an increased maximum hash size depends on the number of rows in the table: for example, if you only have a few rows, a large index hash key would still fit on the index page. No index fragmentation occurs in this case.

When choosing an optimal hash size, consider the data type, the row data, and the database size (especially if a table contains many rows).

The only way to determine if you have chosen an optimal hash size is to run benchmark tests against your UltraLite client application on the target device. Observe how various hash sizes affect the application and query performance, in addition to the changes in database size itself.

Index hashing improves inserts, updates, deletes, and searches when the columns being indexed have a good distribution of values, such as strings that do not have a common prefix, at the cost of bigger index structures. Hashed indexes locate rows first by using the hash, and then by using direct row comparison to differentiate rows with the same hash value. If the hash size is sufficiently big, the hash uniquely identifies a row without reading and comparing the row. However, if the hash size is too big and the page size small, the index may need too many database pages.

## The Data Type

To hash the entire value in a column, note the size required by each data type in the table that follows. UltraLite only uses the maximum hash size if it is necessary, and it never exceeds the maximum hash size you specify. UltraLite uses a smaller hash size if the column type does not use the full byte limit.

Data type	Bytes used to hash the entire value
LONG VARCHAR, DOUBLE, FLOAT, REAL, LONG BINARY, ST_GEOMETRY	Not hashed.
BIT, TINYINT	1
SMALLINT	2
INTEGER, DATE	4
BIGINT, DATETIME, TIME, TIMESTAMP, TIMESTAMP WITH TIME ZONE	8
DECIMAL, NUMERIC	Approximately the precision divided by two.

Data type	Bytes used to hash the entire value
CHAR, VARCHAR	<p>To hash the entire string, the maximum hash size in bytes must match the declared size of the column. In a UTF-8 encoded database, always multiply the declared size by a factor of 2, but only to the allowed maximum of 32 bytes.</p> <p>For example, if you declare a column VARCHAR(10) in a non-UTF-8 encoded database, the required size is 10 bytes. However, if you declare the same column in a UTF-8 encoded database, the size used to hash the entire string is 20 bytes.</p>
BINARY, VARBINARY	<p>The maximum hash size in bytes must match the declared size of the column.</p> <p>For example, if you declare a column BINARY(30), the required size is 30 bytes.</p>
UNIQUEIDENTIFIER	16

For example, if you set a maximum hash size of 6 bytes for a two-column composite index that you declared as INTEGER and BINARY (20) respectively, then based on the data type size requirements, the following changes occur:

- The entire value of the row in the INTEGER column is hashed and stored in the index because only 4 bytes are required to hash integer data types.
- Only the first 2 bytes of the BINARY column are hashed and stored in the index because the first 4 bytes are used by the INTEGER column. If these remaining 2 bytes do not hash an appropriate amount of the BINARY column, increase the maximum hash size.

## The Row Data

The row values of the data being stored in the database also influence the effectiveness of a hashed index.

For example, if you have a common prefix shared among entries of a given column, you can render the hash ineffective if you choose a size that only hashes prefixes. In this case, choose a size that ensures more than just the common prefix is hashed. If the common prefix is long, consider not hashing the values at all.

When a non-unique index stores many duplicate values, and UltraLite cannot hash the entire value, the hash likely cannot improve performance.

## The Database Size

Each index page has some fixed overhead, but the majority of the page space is used by the actual index entries. A larger hash size means each index entry is bigger, which means that fewer entries can fit on a page. For large tables, indexes with large hashes use more pages than indexes with small or no hashes. Requiring more pages increases the database size and degrades performance. The latter typically occurs because the cache can only hold a fixed number of pages, thereby causing UltraLite to swap pages.

The following table gives you an approximation of how the hash size can affect the number of pages required to store data in an index:

Table	Page size	Hash size	Number of entries	Pages required
Table A	4 KB	0	1200	3 pages
Table B	4 KB	32 bytes	116	3 pages
Table C	4 KB	32 bytes	1200 entries	11 pages

## Set the Hash Size

You can set the maximum hash size in two ways:

- To store a database default for the maximum size, set the `max_hash_size` creation parameter when you create your database. If you do not want to hash indexes by default, set this value to 0. Otherwise, you can change it to any value up to 32 bytes, or keep the UltraLite default of 4 bytes.
- Override the default by setting a specific hash size when you create a new index. Use one of the following approaches:
  - In SQL Central, set the Maximum Hash Size property when creating a new index.
  - With SQL, use the `WITH MAX HASH SIZE` clause in either the `CREATE TABLE` or `CREATE INDEX` statement.

## Related Information

[SQL Data Types \[page 288\]](#)

[Adding an UltraLite Index \[page 65\]](#)

[UltraLite `max\_hash\_size` Creation Option \[page 161\]](#)

[CREATE INDEX Statement \[UltraLite\] \[page 531\]](#)

[CREATE TABLE Statement \[UltraLite\] \[page 537\]](#)

### 1.14.2.4 Execution Plans in UltraLite

UltraLite execution plans show how tables and indexes are accessed when a query is executed.

UltraLite includes a **query optimizer**. The optimizer is an internal component of the UltraLite runtime that attempts to produce an efficient plan for the query. It tries to avoid the use of temporary tables to store intermediate results and attempts to ensure that only the pertinent subset of a table is accessed when a query joins two tables.

## Overriding the Optimizer

The optimizer always aims identify the most efficient access plan possible, but this goal is not guaranteed, especially with a complicated query where a great number of possibilities may exist. In extreme cases, you can override the table order it selects by adding the `OPTION (FORCE ORDER)` clause to a query, which forces UltraLite to access the tables in the order they appear in the query. *This option is not recommended for general use.* If performance is slow, a better approach is usually to create appropriate indexes to speed up execution.

### i Note

If you are not going to update data with the query, specify the `FOR READ ONLY` clause in your query. This clause may yield better performance.

### In this section:

#### [Determine the Access Method Used by the Optimizer \[page 576\]](#)

The UltraLite optimizer uses sophisticated optimization strategies when choosing an index for query optimization.

#### [When to View Execution Plans \[page 579\]](#)

An execution plan can give insight into what the database server considers when processing a query.

#### [Viewing an Execution Plan \[page 579\]](#)

Use Interactive SQL to display an UltraLite plan that summarizes how a prepared statement is to be executed. The text plan is displayed in the Interactive SQL Plan Viewer.

#### [How to Read Execution Plans \[page 580\]](#)

UltraLite short plans are textual summaries of how a query is accessed.

## Related Information

[SELECT Statement \[UltraLite\] \[page 558\]](#)

### 1.14.2.4.1 Determine the Access Method Used by the Optimizer

The UltraLite optimizer uses sophisticated optimization strategies when choosing an index for query optimization.

However, with simple queries you cannot easily predetermine which index the optimizer uses to optimize the query performance, or if an index is used at all. As the complexity increases, the index selected depends on the clauses required by your query. Usually, the presence of a `FOR READ ONLY` clause may cause the optimizer to choose a direct table scan instead of an index to yield better query performance.

When optimizing a query, the optimizer looks at the requirements of the query and checks if there are any indexes that it can use to improve performance. If performance cannot be improved with any index, then the



optimizer does not scan one: either a temporary table or a direct page scan is used instead. Therefore, you may need to experiment with your indexes and frequently check the generated execution plans to ensure that:

- You are not maintaining indexes that are not being used by the optimizer.
- You are minimizing the number of temporary tables being created.

For complex queries, knowing which index is used is even less predictable. For example, when a query contains a WHERE predicate and a GROUP BY clause in addition to an ORDER BY clause, one index alone might not satisfy the search conditions of this query. So, if you have created an index to meet the selectivity requirements of the WHERE predicate, you may find that the optimizer does not actually use it. Instead, the optimizer may use an index that offers better performance for the ORDER BY conditions because this clause could require the most processing.

## Checking the Execution Plan

You can check the execution plan either programmatically with the appropriate API call or in the Plan Viewer in Interactive SQL:

### If no index is used

the execution plan appears as follows:

```
scan (T)
```

### If a temporary table is used

the execution plan appears as follows:

```
temp [scan (T) ]
```

### If an index is used

the index name is included the execution plan:

```
scan (T, index_name)
```

### In this section:

#### [Manage Temporary Tables \[page 577\]](#)

A temporary table is used by an access plan to store data during its execution in a transient or temporary work table. This table only exists while the access plan is being executed.

## 1.14.2.4.1.1 Manage Temporary Tables

A temporary table is used by an access plan to store data during its execution in a transient or temporary work table. This table only exists while the access plan is being executed.

The optimizer tries to avoid creating temporary tables to return query results because the entire temporary table must be populated before the first row can be returned. If an index exists, the optimizer tries to use the index first and only creates a temporary table as a last resort.

Generally, temporary tables are used when intermediate results do not fit in the available memory, such as:

- When subqueries need to be evaluated early in the access plan.
- When data in a temporary table is held for a single connection only.
- When a query contains an ORDER BY on a column other than an index.
- When a query contains a GROUP BY on a column other than an index.

It is difficult to anticipate whether an index you have created avoids the necessity for a temporary table. Therefore, check the plans for a query to ensure the indexes you have created are actually being used by the UltraLite query optimizer.

You can avoid using temporary tables by using an index for the columns used in the ORDER BY or GROUP BY clauses.

#### In this section:

##### [Direct Page Scans \[page 578\]](#)

UltraLite uses direct page scans as an alternative to index scans when it is more efficient to access information directly from the database page.

## Related Information

[How to Read Execution Plans \[page 580\]](#)

[When to View Execution Plans \[page 579\]](#)

[UltraLite TEMP\\_DIR Connection Parameter \[page 201\]](#)

### 1.14.2.4.1.1.1 Direct Page Scans

UltraLite uses direct page scans as an alternative to index scans when it is more efficient to access information directly from the database page.

A direct page scan is only used after the optimizer confirms that:

- No preexisting index can return results more efficiently.
- You are not using the query to perform updates. For example, you have declared the statement to be FOR READ ONLY (the default setting if no FOR clause has been specified), or have written the query in such a way that it is obvious that data is not being updated.

Because UltraLite reads the rows directly from the pages on which the rows are stored, query results are returned without order. The order of subsequent query results is unpredictable. If you need the order of rows to be predictable and deterministic, use an ORDER BY clause to get results in a consistent order. However, if order is not important, you can omit the ORDER BY clause to improve query performance.

#### **i** Note

You cannot use direct page scans if you are using a UTable class in an UltraLite API to program your application.

You can check to see when UltraLite scans a page directly or which index was used to return results.

## Related Information

[Determine the Access Method Used by the Optimizer \[page 576\]](#)

[Index Scan Creation and Maintenance \[page 570\]](#)

### 1.14.2.4.2 When to View Execution Plans

An execution plan can give insight into what the database server considers when processing a query.

View an execution plan in Interactive SQL when you need to know:

- What index will be used to return the results. An index scan object contains the name of the table and the index on that table that is being used.
- Whether a temporary table will be used to return the results. Temporary tables are written to the UltraLite temporary file.
- Which order tables are joined. This information allows you to determine how performance is affected.
- Why a query is running slowly or to ensure that a query does not run slowly.

## Related Information

[UltraLite TEMP\\_DIR Connection Parameter \[page 201\]](#)

### 1.14.2.4.3 Viewing an Execution Plan

Use Interactive SQL to display an UltraLite plan that summarizes how a prepared statement is to be executed. The text plan is displayed in the Interactive SQL Plan Viewer.

## Context

In UltraLite, an execution plan is strictly a short textual summary of the plan. No other plan types are supported. However, being a short plan, it allows you to compare plans quickly, because information is summarized on a single line.

## Procedure

1. Click **Tools > Plan Viewer**.
2. In the *SQL* pane, type a query.
3. Click *Get Plan* to generate a plan for the specified SQL statements.

## Results

The text plan appears in the lower pane of the Plan Viewer.

## Example

Consider the following statement:

```
SELECT I.inv_no, I.name, T.quantity, T.prod_no
FROM Invoice I, Transactions T
WHERE I.inv_no = T.inv_no
```

This statement might produce the following plan:

```
join[scan(Invoice,primary),index-scan(Transactions,secondary)]
```

The plan indicates that the join operation is completed by reading all rows from the Invoice table (following an index named primary). It then uses the index named secondary from the Transactions table to read only the row whose inv\_no column matches.

## Related Information

[How to Read Execution Plans \[page 580\]](#)

[Interactive SQL Utility \(dbisql\)](#)

### 1.14.2.4.4 How to Read Execution Plans

UltraLite short plans are textual summaries of how a query is accessed.

You must understand how the operations of either a join or a scan of a table are implemented to read a short plan.

#### For scan operations

Represented with a single operand, which applies to a single table only and uses an index. The table name and index name are displayed as round brackets ( (, ) ) following the operation name.

## For other operations

Represented with one or more operands, which can also be plans in and of themselves. In UltraLite, these operands are comma-separated lists contained by square brackets ( [ ] ).

## Operation list

Operations supported by UltraLite are listed in the table that follows.

Operation	Description
<i>count</i> (*)	Counts the number of rows in a table.
<i>distinct</i> [ plan ]	Implements the DISTINCT aspect of a query to compare and eliminate duplicate rows. It is used when the underlying plan sorts rows in such a way that duplicate contiguous rows are eliminated. If two contiguous rows match, only the first row is added to the result set.
<i>dummy</i>	No operation performed. It only occurs in two cases: <ul style="list-style-type: none"><li>• When you specify DUMMY in a FROM clause.</li><li>• When the FROM clause is missing from the query.</li></ul>
<i>filter</i> [ plan ]	Executes a search condition for each row supplied by the underlying plan. Only the rows that evaluate to true are forwarded as part of the result set.
<i>group-by</i> [ plan ]	Creates an aggregate of GROUP BY results, to sort multiple rows of grouped data. Rows are listed in the order they occur and are grouped by comparing contiguous rows.
<i>group-single</i> [ plan ]	Creates an aggregate of GROUP BY results, but only when it is known that a single row will be returned.
<i>keyset</i> [ plan ]	Records which rows were used to create rows in a temporary table so UltraLite can update the original rows. If you do not want those rows to be updated, then use the FOR READ ONLY clause in the query to eliminate this operation.
<i>index-scan</i> ( table-name, index-name )	Reads only part of the table; the index is used to find the starting row.
<i>join</i> [ plan, plan ]	Performs an inner join between two plans.
<i>lojoin</i> [ plan, plan ]	Performs a left outer join between two plans.
<i>like-scan</i> ( table-name, index-name )	Reads only part of a table; the index is used to find the starting row by pattern matching.

Operation	Description
<code>rowlimit[ plan ]</code>	Performs the row limiting operation on propagated rows. Row limits are set by the TOP n or FIRST clause of the SELECT statement.
<code>scan( table-name, index-name )</code>	Reads an entire table following the order indicated by the index.
<code>sub-query[ plan ]</code>	Marks the start of a subquery.
<code>temp[ plan ]</code>	Creates a temporary table from the rows in the underlying plan. UltraLite uses a temporary table when underlying rows must be ordered and no index was found to do this ordering.  You can add an index to eliminate the need for a temporary table. However, each additional index used increases the duration needed to insert or synchronize rows in the table for which the index applies.
<code>union-all[ plan, ..., plan ]</code>	Performs a UNION ALL operation on the rows generated in the underlying plan.

## 1.14.3 Insert and Update Performance Tips

Several topics are provided to help improve performance of inserts and updates in UltraLite databases.

### In this section:

[Transaction and Row State Management \[page 582\]](#)

UltraLite maintains state information along with the data in the database.

[Flush Single or Grouped Transactions \[page 586\]](#)

You can choose your recovery point in UltraLite by delaying committed transaction flushes.

### 1.14.3.1 Transaction and Row State Management

UltraLite maintains state information along with the data in the database.

UltraLite tracks and stores state information so it can manage:

- Concurrent connections, so UltraLite can share resources as required.
- Synchronization progress state, to ensure that synchronization occurs successfully.
- Row state, to maintain data integrity by tracking how data has changed between synchronizations.
- Transactions, to determine when and how data gets committed. In UltraLite, a transaction is processed in its entirety or not at all.
- Recovery and backup information, to protect data against operating system crashes, and end-user actions such as removing storage cards, or device resets while UltraLite is running.

## In this section:

### [UltraLite Concurrency \[page 583\]](#)

There are several aspects to UltraLite concurrency.

### [UltraLite Database Row State Management \[page 584\]](#)

UltraLite maintains row state information. Tracking the state of tables and rows is particularly important for data synchronization.

### [UltraLite Transaction Processing \[page 585\]](#)

A **transaction** is a logical set of operations that are executed atomically: either all operations in the transaction are stored in the database or none are.

## Related Information

[UltraLite Synchronization Client Features \[page 10\]](#)

[UltraLite Database Back up and Recovery \[page 52\]](#)

## 1.14.3.1.1 UltraLite Concurrency

There are several aspects to UltraLite concurrency.

### **Multiple UltraLite database access**

A single application can open connections to multiple databases.

### **Multiple applications**

An UltraLite database can only be opened by one process at a time. Use the UltraLite engine to handle multiple applications.

### **Multiple threads**

UltraLite supports multithreaded applications. A single application can be written to use multiple threads, each of which can connect to the same or different databases. Only a single thread may access a given connection (or SQLCA) at a time. (UltraLite is thread safe and connections are thread agnostic, but you may not make concurrent requests on a single connection/SQLCA.) Requests on a given database are generally serialized; synchronization is an exception. Requests on separate databases run concurrently.

### **Multiple transactions/requests**

Each connection can have a single transaction in progress at one time. Transactions can consist of a single request or multiple requests. Data modifications made during a transaction are not made permanent in the database until the transaction is committed. Either all data modifications made in a transaction are committed, or all are rolled back.

### **Synchronization**

During upload and download, read-write access to the database is permitted. However, if an application changes a row that the download then attempts to change, the download fails and rolls back. Use the Disable Concurrency synchronization parameter to disable access to data during synchronization.

UltraLite supports resumable downloads on all platforms. Use resumable downloads to complete downloads that stop because of network problems or because your application must exit.

## Related Information

[Failed Downloads](#)

[UltraLite Database Limitations \[page 15\]](#)

[UltraLite Transaction Processing \[page 585\]](#)

[UltraLite Clients \[page 73\]](#)

[Additional Parameters Synchronization Parameter \[page 95\]](#)

### 1.14.3.1.2 UltraLite Database Row State Management

UltraLite maintains row state information. Tracking the state of tables and rows is particularly important for data synchronization.

An internal marker is used to keep track of the row state in an UltraLite database. Row states control transaction processing, recovery, and synchronization. When an application inserts, updates, or deletes a row, UltraLite modifies the state of the row to reflect the operation and the connection that performed the operation. When a transaction is committed, the states of all rows affected by the transaction are modified to reflect the commit. If an unexpected failure occurs during a commit, the entire transaction is rolled back. The following list summarizes these behaviors:

#### **When a delete is issued**

The state of each affected row is changed to reflect the fact that it was deleted. When a delete is undone through a rollback, the original state of the row is restored.

#### **When a delete is committed**

The affected rows are not always removed from memory. If the row has never been synchronized, then it is removed. If the row has been synchronized, then it is not removed, because the delete operation needs to be synchronized to the consolidated database first. After the next synchronization, the row is removed from memory.

#### **When a row is updated**

A new version of the row is created. The states of the old and new rows are set so the old row is no longer visible and the new row is visible.

#### **When a row update is committed**

When a transaction is committed, the states of all rows affected by the transaction are modified to reflect the commit. When an update is synchronized, both the old and new versions of the row are needed to allow conflict detection and resolution. The old row is then deleted from the database and the new row simply becomes a normal row.

#### **When a row is added**

The row is added to the database and is marked as not committed.

#### **When an added row is committed**



The row is marked as committed and is also flagged as requiring synchronization with the consolidated database.

## Related Information

[UltraLite Database Back up and Recovery \[page 52\]](#)

[Flush Single or Grouped Transactions \[page 586\]](#)

[UltraLite Transaction Processing \[page 585\]](#)

### 1.14.3.1.3 UltraLite Transaction Processing

A **transaction** is a logical set of operations that are executed atomically: either all operations in the transaction are stored in the database or none are.

An UltraLite application's access to the UltraLite runtime is serialized. While it is possible for multiple transactions to be open simultaneously, UltraLite only processes one transaction at a time. This behavior means that an application cannot:

- Have blocked transactions (also known as deadlocks). UltraLite never blocks a request based on an existing row lock. In this case, UltraLite immediately returns an error.
- Overwrite outstanding changes. A transaction cannot overwrite another transaction's outstanding changes. When a transaction changes a row, UltraLite locks that row until the transaction is **committed** or **rolled back**. The lock prevents other transactions from changing the row, although they can still read the row.

#### i Note

All UltraLite APIs except the UltraLiteJ and C++ APIs can operate in **autocommit** mode.

In autocommit mode, UltraLite executes a commit after each operation. Some APIs use autocommit by default. If you are using one of these interfaces, you must set autocommit to off to exploit multi-operation transactions. The way of turning autocommit off depends on the programming interface you are using. In most interfaces it is a property of the connection object.

For example, two applications, A and B, are reading the same row from the database and they both calculate new values for one of its columns based on the data they read. If A updates the row with its new value and B then tries to modify the same row, B gets an error. An attempt to change a locked row sets the error SQLCODE SQLE\_LOCKED, while an attempt to change a deleted row sets the error SQLE\_NOTFOUND. Therefore, you should program your application so it checks the SQLCODE value after attempting to modify data.

## Related Information

[Error Handling \[page 665\]](#)

[Error Handling in UltraLite.NET \[page 619\]](#)

[Transaction Management in UltraLite C++ \[page 664\]](#)

[Transaction Management in UltraLite.NET \[page 618\]](#)

## 1.14.3.2 Flush Single or Grouped Transactions

You can choose your recovery point in UltraLite by delaying committed transaction flushes.

When UltraLite releases the commit to storage, the recovery point helps control when a subset of SQL statements in a transaction triggers additional operational overhead.

By default, UltraLite uses an operational-based default that flushes individual transactions to storage immediately upon a commit. For some deployments, these frequent operations can be excessive and limit the amount of transaction throughput. To reduce the performance expense caused by this default, you may choose a state-based approach. Especially for applications that rely on autocommit operations, this approach delays the additional overhead required to flush the committed transactions to storage:

### **On checkpoint**

You can set your own checkpoint, and then use it to release the work performed over the course of time. You can use as many checkpoints as you require, either within a single transaction or over multiple transactions.

### **Grouped**

You can choose a transaction count threshold and/or a timeout threshold to release the work performed.

Delaying commit flushes based on state yields better performance and a cleaner application design because applications are not required to wait for a response from UltraLite. By delaying commit flushes you also minimize the exposure to transactions by giving more granular control over data for which work has not been fully completed. For example, in a sales application, an order may be available to a second application before all items have been added or even approved.

However, it is important for you to take into account the recoverability of a transaction for which commit flushes have been delayed. Transactions that have not been released cannot be recovered. Therefore, you need to evaluate the trade-off between the data integrity of your application and its performance.

## **Related Information**

[UltraLite COMMIT\\_FLUSH Connection Parameter \[page 188\]](#)

[UltraLite commit\\_flush\\_count Option \[Temporary\] \[page 208\]](#)

[UltraLite commit\\_flush\\_timeout Option \[Temporary\] \[page 209\]](#)

[CHECKPOINT Statement \[UltraLite\] \[page 529\]](#)

## 1.14.4 UltraLite Benchmark Tips

Benchmark testing activity is generally performed before reaching the production stage of the application development cycle.

This phase requires that you test the UltraLite database with the application and ensure both components inter-operate as efficiently as possible. If you discover through your tests that performance is not as efficient as it could be, you can then tune your database and/or optimize your application to improve benchmark results.

### i Note

If your UltraLite deployment is part of a MobiLink synchronization environment, remember to test synchronization performance as well.

#### In this section:

##### [Types of Benchmark Tests \[page 587\]](#)

There are several kinds of performance you can observe using benchmark tests.

##### [Methodology \[page 589\]](#)

There are three phases used for benchmark testing: preparation, creation, and execution.

## Related Information

[MobiLink Tuning for Performance](#)

### 1.14.4.1 Types of Benchmark Tests

There are several kinds of performance you can observe using benchmark tests.

- SQL statements: The UltraLite database has been optimized to handle SQL queries efficiently and return results quickly. Nonetheless, you should see how well important queries perform to improve database performance.
- Synchronization: The key to achieving optimal MobiLink synchronization throughput is to have multiple synchronizations occurring simultaneously and executing efficiently.
- Indexes
- Table design
- Application code
- Device configuration: For example, compare using external flash memory and a device's internal memory as deployment locations for UltraLite.
- Database configuration: For example, try different cache sizes, page sizes, reserve sizes, indexes, hash sizes, and so on.
- Data throughput (based on transactions per second): While UltraLite is not typically a database intended for mass data entry processing, depending on your business requirements, this benchmark is one that you may to run tests for.

- Software changes: You should test the effect of software changes between two different versions of UltraLite or different versions of an application.

**In this section:**

[SQL Query Testing \[page 588\]](#)

There are two types of query benchmark testing you can perform: representational SQL benchmarks, and targeted SQL benchmarks.

## Related Information

[MobiLink Tuning for Performance](#)

### 1.14.4.1.1 SQL Query Testing

There are two types of query benchmark testing you can perform: representational SQL benchmarks, and targeted SQL benchmarks.

#### Representational SQL Benchmarks

This type of testing requires that you test a selection of statements that are representative of typical transactions that the application performs during day-to-day operations. Different applications require different benchmark tests because the fundamental business use for each application varies. For example, a meter reading application might simply test a basic INSERT statement. However a mobile sales force application might test multiple INSERT statements, in addition to multiple SELECT statements and perhaps an UPDATE statement.

The volume of queries in your application can limit what you can realistically test. If you have excessive query processing, you may be limited to performing the targeted SQL benchmark testing.

#### Targeted SQL Benchmarks

If you have a lot of statements used by your applications, you may want to narrow the scope of your test to include some or all of the following:

- The most frequently used statements.
- The statements that process high volumes of data.
- The statements that have time-sensitive requirements.
- The statements that are most important to the business case of your application.
- The most complex statements: for example, those that have the largest number of table joins or that use many subqueries. These types of statements can use a large amount of device resources. Even if the

statements are only used infrequently, you may want to check that they don't exceed the capacity of the device.

- The statements that are not supported by an index.
- The statements that use a large amount of memory resources.

## 1.14.4.2 Methodology

There are three phases used for benchmark testing: preparation, creation, and execution.

### 1. The preparation phase

Allows you to finalize your database design and reach a stable point in your application development before starting your benchmark testing.

### 2. The creation phase

Allows you to build a custom program that replicates the end-user behavior you predict for your UltraLite deployment.

### 3. The execution and analysis phase

Allows you to fine tune different elements of your database and record the results of those changes so you can analyze them. Tests are repeated until the maximum benefit of all modifications has been reached.

#### In this section:

##### [The Preparation Phase \[page 589\]](#)

The preparation phase allows you to get your database and application in a state where they can be successful benchmark candidates, and to determine what goals you hope to achieve from your tests.

##### [The Creation Phase \[page 590\]](#)

Create tests that yield reliable results. Otherwise, you cannot legitimately compare results over time.

##### [Performing a Benchmark Test \[page 591\]](#)

Perform a benchmark test.

## 1.14.4.2.1 The Preparation Phase

The preparation phase allows you to get your database and application in a state where they can be successful benchmark candidates, and to determine what goals you hope to achieve from your tests.

In the preparation phase, you do the following:

1. Complete the logical design of the database.  
Ensure you have:
  - Created and populated tables with representative data.
  - Created indexes to retrieve data in those tables more effectively.
2. Prepare the physical deployment environment of both the database and application. The deployment environment must accurately represent the final production environment: that is, the lab and production environments should share the same memory and disk configurations on the same platform/device type.
3. Ensure you have reached a stable point in your application programming phase. Remember you are looking for performance optimizations not defects; however, the latter may also be revealed as a result of testing.

All database queries should access and return as much data as required. If the production environment requires sorting of data, ensure queries include this data as well. Otherwise, the application cannot accurately test representative memory requirements.

4. Deploy a copy of the database and application to the final disk location.
5. Decide what element of the database performance you want to check and potentially tune.

You can now run benchmark tests against your database and application.

## Related Information

[Types of Benchmark Tests \[page 587\]](#)

### 1.14.4.2.2 The Creation Phase

Create tests that yield reliable results. Otherwise, you cannot legitimately compare results over time.

The following characteristics make a benchmark test effective and reliable:

#### Goal

Are you looking to capture a performance ratio or are you trying to see the duration required to process a command against a database? For the former, if you are testing SQL performance, you may want to run one or more statements repeatedly until a set time interval has expired. This testing gives you the throughput ratio, which can be summarized as follows:

```
statement-number / time-interval = throughput ratio
```

#### Environment

Establish a test environment as your baseline and record the design and scope of it. If you cannot run the same test under the same conditions, you cannot legitimately compare results of that test. Additionally, the hardware and software you use in the lab as part of your benchmark test should match that of your production environment.

#### State

Reliable benchmark tests always start each iteration with the same action. Decide whether third-party applications should operate concurrently with UltraLite. If they affect performance, you should add them to the benchmark test design. For third-party applications that should not be running, always exit these applications completely, even minimized or idle applications/processes could skew results because memory is still being used.

#### Results

Results of benchmarks must be captured in a consistent way after each iteration of the test. Over time the results can indicate a trend and help you determine what changes can yield an improvement in UltraLite performance, either in the database or in the application (or both).

#### Timing mechanism

Benchmark tests simulate user actions; therefore, you typically track the elapsed execution times of these actions. Ensure your timing mechanism is systematic so execution times are accurately reflected in the results of your tests.

## Related Information

[SQL Query Testing \[page 588\]](#)

### 1.14.4.2.3 Performing a Benchmark Test

Perform a benchmark test.

#### Context

Benchmark testing is the phase during which you tune your database by iteratively running a test, modifying something about the database (for example, the value of one or more database properties or connection parameters), and then running the test again to see the outcome of any changes.

The following procedure assumes you are testing different database properties and/or connection parameters to find the maximum benefit. Repeat this procedure until all parameters that require testing have been tested.

#### i Note

Choose only those properties or parameters that are significant to the workload and the objectives of your UltraLite deployment.

#### Procedure

1. Create a baseline by running the first iteration of the test. In this case, because you are testing different database properties and/or connection parameters, you would use UltraLite defaults wherever possible.
2. Begin your normal test runs by tuning only one database property or connection parameter at a time. This limitation ensures that the results you collect are systematic in their approach and helps you more readily determine when you have reached the maximum benefit of your tuning activities.
3. Output from the benchmark program should include:
  - an identifier or label for each test
  - the iteration of the program execution
  - the name of the element being checked and what you did to change it
  - the recorded elapsed time

For example, even though you could test other database parameters, if you limited your test to just varying page sizes, cache sizes, and reserve sizes, your output might be saved to a table that looks similar to the example that follows:

PROP/PARM	VALUES		
TEST NUMBER	001	002	003
page_size	1	2	8
CACHE_SIZE	128	256	512

RESERVE_SIZE	128	256	512
STMT ID	EXECUTION (seconds)		
01	01.55	01.50	01.49
02	02.01	02.20	01.59
03	00.33	00.55	00.44

- When you have completed an iteration, return the database to the baseline state to ensure you do not inadvertently contaminate results of subsequent runs.

## Results

Depending on the results of the benchmark test, do one of the following:

- If performance improves, change the value of the same property or parameter and rerun the test. Keep tuning this value until you cannot improve performance any further.
- If the performance worsens, return the value of the property or parameter to the previous value.

## Next Steps

Test another new property or parameter.

## 1.15 UltraLite Troubleshooting

Several topics are provided to help you troubleshoot problems with your UltraLite database.

### In this section:

#### [Unable to Start the UltraLite Engine \[page 593\]](#)

You have to use the START connection parameter to start the UltraLite engine with the following definition; however, the client returns SQLE\_UNABLE\_TO\_CONNECT\_OR\_START.

#### [Unable to Connect to Databases After Upgrade \[page 593\]](#)

You have upgraded UltraLite. You discover that you are able to create an empty database using the administration tools. However, when you try to connect to this or any other database (including `CustDB.udb`) with SQL Central, you receive an error. Connecting to SQL Anywhere databases works without incident, however.

#### [UltraLite Database Corruption \[page 594\]](#)

You receive an error message indicating that the database is corrupted.

#### [Database Size Not Stabilizing \[page 595\]](#)

Your application collects a lot of large binary objects among multiple client devices, synchronizes this information to a consolidated database, and then the synchronized data is deleted from each client device. However, the database size remains large despite the data being removed from the database.

#### [Importing ASCII Data into a New UltraLite Database \[page 596\]](#)



You have created a new UltraLite database, but have a .CSV ASCII data file that you cannot import.

#### [Utilities Still Running as the Previous Version \[page 597\]](#)

You have just installed UltraLite 17. However, when you try to run any of the UltraLite utilities, the previous version starts.

#### [Result Set Changes Unpredictably \[page 598\]](#)

You run a query and the result set you expect changes each time you run it.

#### [UltraLite Engine Client Fails with Error -764 \[page 598\]](#)

You are running the UltraLite engine on Microsoft Windows Mobile device, and the client returns a -764 error.

## 1.15.1 Unable to Start the UltraLite Engine

You have to use the START connection parameter to start the UltraLite engine with the following definition; however, the client returns SQLE\_UNABLE\_TO\_CONNECT\_OR\_START.

```
START="\Program Files\uleng17.exe"
```

### Explanation

The location of the quotes is incorrect.

### Recommendation

For this parameter to work, the first quotation mark must follow the \ character. For example, you can delimit spaces in this path as follows:

```
START="Program Files\uleng17.exe"
```

or

```
START="'Program Files\uleng17.exe''
```

## 1.15.2 Unable to Connect to Databases After Upgrade

You have upgraded UltraLite. You discover that you are able to create an empty database using the administration tools. However, when you try to connect to this or any other database (including CustDB.udb)

with SQL Central, you receive an error. Connecting to SQL Anywhere databases works without incident, however.

## Explanation

You did not close all SQL Anywhere applications and processes. Therefore, your UltraLite plug-ins were not installed correctly.

## Recommendation

Remove and reinstall SQL Anywhere.

1. Close SQL Central, Interactive SQL, and any running database engines.
2. Run the following commands:

```
dbisql -terminate
```

```
scjview -terminate
```

3. Open the Windows *Task Manager*, and end any `scjview.exe` and `dbisql.exe` processes.
4. Reinstall the latest version of UltraLite.

## Related Information

[UltraLite Upgrades](#)

## 1.15.3 UltraLite Database Corruption

You receive an error message indicating that the database is corrupted.

### Symptom

Your UltraLite database may be corrupt if it:

- Generates the following errors:
  - SQLE\_DEVICE\_ERROR
  - SQLE\_DATABASE\_ERROR (can also be a symptom of other issues)
  - SQLE\_MEMORY\_ERROR (can also be a symptom of other issues)

- Crashes or returns invalid query results.

## Explanation

There are two more typical causes corruption:

- The more frequent cause occurs if the device has problems storing the file, thereby spuriously changing the contents of it. This issue usually stops the UltraLite database from functioning fairly quickly.
- The less frequent cause occurs if an error in the UltraLite code fails to maintain an index correctly. These issues can go undetected for much longer because the change to the results of a query are more subtle.

## Recommendation

Checksums are used to detect offline corruption in an UltraLite database, which can help reduce the chances of other data being corrupted as the result of a bad critical page. If a checksum validation fails when the UltraLite database loads a page, then UltraLite immediately stops the database and reports a fatal error. This error cannot be corrected. Instead you must:

1. Report the error. It is helpful if you know the sequence of events that caused the corruption to occur, and if the error is reproducible.
2. If you need the data, unload the contents of the UltraLite database to a file.
3. Create a new UltraLite database.
4. Repopulate the data either by synchronizing or by loading the unloaded data.

## Related Information

[UltraLite checksum\\_level Creation Option \[page 149\]](#)

### 1.15.4 Database Size Not Stabilizing

Your application collects a lot of large binary objects among multiple client devices, synchronizes this information to a consolidated database, and then the synchronized data is deleted from each client device. However, the database size remains large despite the data being removed from the database.

## Explanation

This is a concern because file size needs to be managed carefully due to limited resources of the device.

Database size should only increase if your data grows in the database. However, once grown, the database file keeps that size, and does not shrink on its own. Free space is maintained internally to the file.

## Recommendation

Ensure you are not using the STOP SYNCHRONIZATION DELETE or TRUNCATE statements for tables that do not get synchronized. Instead use the DELETE statement with a FROM `table-name` clause for tables that do not get synchronized.

Recreate the database post-synchronization:

1. Create the database that is deployed to the devices.
2. Creating a SQL script of DDL statements that define the schema required by the client devices.
3. Synchronize the data.
4. Drop the database.
5. Create a new, empty database and use standard database schema with the ALTER DATABASE SCHEMA FROM FILE statement.

## Related Information

[Deploying UltraLite Database Schema Upgrades \[page 131\]](#)

[STOP SYNCHRONIZATION DELETE Statement \[UltraLite\] \[page 562\]](#)

[TRUNCATE TABLE Statement \[UltraLite\] \[page 565\]](#)

[DELETE Statement \[UltraLite\] \[page 544\]](#)

[ALTER DATABASE SCHEMA FROM FILE Statement \[UltraLite\] \[page 520\]](#)

## 1.15.5 Importing ASCII Data into a New UltraLite Database

You have created a new UltraLite database, but have a `.csv` ASCII data file that you cannot import.

### Explanation

The `.csv` format is not supported by any of the UltraLite administration tools.

## Recommendation

You can try one of the following techniques:

- Use Interactive SQL (dbisql) to import the data. You can connect to the UltraLite database and then click [▶ Data ▶ Import Data ▾](#). Alternatively, you can connect to the UltraLite database and run the INPUT statement (this statement cannot be used in an UltraLite PreparedStatement object).

### i Note

UltraLite requires primary keys. Although Interactive SQL can create the table for you, it does not automatically create the primary keys for them. Always connect to an empty UltraLite database you have created for this purpose.

- If you incorporate this functionality as part of a batch process, you must write your own code.

## Related Information

[INPUT Statement \[Interactive SQL\]](#)

[Interactive SQL for UltraLite Utility \(dbisql\) \[page 214\]](#)

## 1.15.6 Utilities Still Running as the Previous Version

You have just installed UltraLite 17. However, when you try to run any of the UltraLite utilities, the previous version starts.

### Explanation

If you have multiple versions of UltraLite on your computer, you must pay attention to your system path when using the administration. Since the installation adds the most recently installed version executable directory to the end of your system path, it is possible to install a new version of the software, and still inadvertently be running the previously installed version.

## Recommendation

There are various workarounds to this problem.

## Related Information

[How to Ensure That You Are Running the Correct Version of the Utilities When You Have Multiple Versions Installed](#)

### 1.15.7 Result Set Changes Unpredictably

You run a query and the result set you expect changes each time you run it.

#### Explanation

Carefully review the result set you are getting. Are the results in the set truly different? Or are they simply being returned in the most efficient order each time. The order selected can change each time you execute the query, depending on when you last accessed the row and other factors.

#### Recommendation

If your result set must be returned in a predictable or consistent order, ensure that the SELECT statement includes an ORDER BY clause. If the result set is still returning results incorrectly, your database may be corrupt.

## Related Information

[SELECT Statement \[UltraLite\] \[page 558\]](#)

[UltraLite Database Corruption \[page 594\]](#)

### 1.15.8 UltraLite Engine Client Fails with Error -764

You are running the UltraLite engine on Microsoft Windows Mobile device, and the client returns a -764 error.

#### Applies to

Microsoft Windows Mobile

## Explanation

An error of -764 means that the engine could not be found and was unable to start.

## Recommendation

Consider one of the following actions:

- Consider redeploying the engine to the recommended deployment location, the `\Windows` directory. UltraLite automatically looks for the engine files in this location.
- If you have install the engine to any other location, ensure your connection code uses the `START` connection parameter.
- If you have used the `START` connection parameter, and you are sure the path to the engine is correct, ensure you have used the correct escape sequences for special characters in the path name. For example, you may need to change this code:

```
ULConnection conn = new ULConnection(@"dbf=\Program Files\HelloEngine
\HelloEngine.udb;
START=\Windows\uleng17.exe")
```

To something similar to:

```
ULConnection conn = new ULConnection(@"dbf=\\\\"Program Files \\\HelloEngine\
\HelloEngine.udb;
START=\\Windows\\uleng17.exe");
```

## Related Information

[UltraLite Engine Startup \[page 133\]](#)

[UltraLite START Connection Parameter \[page 200\]](#)

## 2 UltraLite.NET Application Development

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

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

### i Note

You can use the UltraLite C++ API as an alternative to the UltraLite.NET API to create applications for Microsoft Windows Mobile devices and desktop.

#### In this section:

##### [UltraLite .NET System Requirements and Supported Platforms \[page 601\]](#)

UltraLite.NET supports Microsoft Windows Mobile devices. Third-party software is required for database development.

##### [SQL Anywhere Tools in Microsoft Visual Studio \[page 602\]](#)

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

##### [Connection Setup for an UltraLite Database \[page 602\]](#)

UltraLite applications must connect to a database before carrying out operations on the data in it.

##### [Data Creation and Modification in UltraLite.NET Using SQL Statements \[page 604\]](#)

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

##### [Data creation and modification in UltraLite.NET using the UTable Class \[page 610\]](#)

UltraLite applications can access table data using SQL statements or by using the UTable class.

##### [Transaction Management in UltraLite.NET \[page 618\]](#)

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.

##### [Schema Information in UltraLite.NET \[page 618\]](#)

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

##### [Error Handling in UltraLite.NET \[page 619\]](#)

You can use the standard .NET error-handling features to handle errors. Most UltraLite methods throw ULErrorException errors.

##### [MobiLink Data Synchronization in UltraLite.NET \[page 620\]](#)

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

##### [How to Deploy UltraLite.NET Applications \[page 621\]](#)



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

[Tutorial: Building a Microsoft Windows Mobile Application Using UltraLite.NET \[page 625\]](#)

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

## Related Information

[UltraLite C++ Application Development \[page 644\]](#)

[UltraLite .NET API Reference](#)

## 2.1 UltraLite .NET System Requirements and Supported Platforms

UltraLite.NET supports Microsoft Windows Mobile devices. Third-party software is required for database development.

### Development Platforms

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

- A supported desktop version of Microsoft Windows.
- Microsoft Visual Studio.

### Target Platforms

UltraLite.NET supports the following target platforms:

- Microsoft .NET Framework 2.0 or later for Microsoft Windows.
- Microsoft .NET Compact Framework 2.0 or later for Microsoft Windows Mobile.

## Related Information

[Supported Platforms](#)

## 2.2 SQL Anywhere Tools in Microsoft Visual Studio

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

## 2.3 Connection Setup for an UltraLite Database

UltraLite applications must connect to a database before carrying out operations on the data in it.

### i Note

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

### Using the ULConnection Object

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

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

#### **Commit behavior**

By default, UltraLite.NET applications are in AutoCommit mode. Each Insert, Update, or Delete statement is committed to the database immediately. You can use ULConnection.BeginTransaction to define the start of a transaction in your application.

#### **User authentication**

You can change the user ID and password for the application from the default values of DBA and sql, respectively, by using methods. Each UltraLite database can define a maximum of four user IDs.

#### **Synchronization**

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

#### **Tables**

UltraLite tables are accessed using methods of the Connection object.

#### **Commands**

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

### Multithreaded Applications

Each ULConnection object and all objects created from it should be used on a single thread. If your application requires multiple threads accessing the UltraLite database, each thread requires a separate connection. For

example, if you design your application to perform synchronization in a separate thread, you must use a separate connection for the synchronization and you must open the connection from that thread.

#### In this section:

[Connecting to an UltraLite Database Using UltraLite.NET \[page 603\]](#)

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

## Related Information

[Transaction Management in UltraLite.NET \[page 618\]](#)

[MobiLink Data Synchronization in UltraLite.NET \[page 620\]](#)

[Data creation and modification in UltraLite.NET using the ULTable Class \[page 610\]](#)

[Data Creation and Modification in UltraLite.NET Using SQL Statements \[page 604\]](#)

## 2.3.1 Connecting to an UltraLite Database Using UltraLite.NET

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

### Procedure

1. Declare a `ULConnection` object.

```
ULConnection conn;
```

2. Open a connection to an existing database.

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

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

### Results

A connection to the `mydata.udb` database is established.

## Next Steps

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

## 2.4 Data Creation and Modification in UltraLite.NET Using SQL Statements

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

The following tasks can be performed using SQL statements:

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

### In this section:

#### [Data Modification in UltraLite.NET Using INSERT, UPDATE, and DELETE \[page 605\]](#)

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

#### [Retrieving Data in UltraLite.NET Using SELECT \[page 608\]](#)

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

#### [Result Set Schema Description \[page 609\]](#)

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

#### [SQL Result Set Navigation in UltraLite.NET \[page 609\]](#)

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

## Related Information

[Data creation and modification in UltraLite.NET using the ULTable Class \[page 610\]](#)  
[SQL Statements](#)

## 2.4.1 Data Modification in UltraLite.NET Using INSERT, UPDATE, and DELETE

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

### In this section:

#### [Inserting a Row in a Table Using UltraLite.NET \[page 605\]](#)

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

#### [Updating a Row in a Table Using UltraLite.NET \[page 606\]](#)

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

#### [Deleting a Row in a Table in UltraLite.NET \[page 607\]](#)

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

### 2.4.1.1 Inserting a Row in a Table Using UltraLite.NET

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

#### Procedure

1. Declare a `ULCommand`.

```
ULCommand cmd;
```

2. Assign a SQL statement to the `ULCommand` object.

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

3. Assign input parameter values for the statement.

The following code shows a string parameter.

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

4. Execute the statement.

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

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

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

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

## Results

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

### 2.4.1.2 Updating a Row in a Table Using UltraLite.NET

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

## Procedure

1. Declare a ULCommand.

```
ULCommand cmd;
```

2. Assign a statement to the ULCommand object.

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

3. Assign input parameter values for the statement.

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

4. Execute the statement.

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

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

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

## Results

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

### 2.4.1.3 Deleting a Row in a Table in UltraLite.NET

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

#### Procedure

1. Declare a ULCommand.

```
ULCommand cmd;
```

2. Assign a statement to the ULCommand object.

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

3. Assign input parameter values for the statement.

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

4. Execute the statement.

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

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

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

## Results

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

## 2.4.2 Retrieving Data in UltraLite.NET Using SELECT

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

### Procedure

1. Declare a ULCommand object to holds the query.

```
ULCommand cmd;
```

2. Assign a statement to the object.

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

3. Execute the statement.

Query results can be returned as one of several types of objects. In this example, a ULDataReader object is used.

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

### Results

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



## 2.4.3 Result Set Schema Description

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

### Example

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

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

## 2.4.4 SQL Result Set Navigation in UltraLite.NET

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

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

#### **MoveAfterLast**

moves to a position after the last row.

#### **MoveBeforeFirst**

moves to a position before the first row.

#### **MoveFirst**

moves to the first row.

#### **MoveLast**

moves to the last row.

#### **MoveNext**

moves to the next row.

#### **MovePrevious**

moves to the previous row.

#### **MoveRelative(offset)**

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

## 2.5 Data creation and modification in UltraLite.NET using the ULTable Class

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

The following tasks can be performed using the Table API:

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

### In this section:

#### [Row Navigation in UltraLite.NET \[page 611\]](#)

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

#### [UltraLite Modes \[page 612\]](#)

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

#### [Row Insertion in UltraLite.NET \[page 612\]](#)

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.

#### [Row Updates \[page 613\]](#)

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.

#### [Row Searches \[page 614\]](#)

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.

#### [Row Retrieval \[page 616\]](#)

Row operations can be managed using various methods in the ULTable class.

#### [Row Deletions in UltraLite.NET \[page 617\]](#)

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.

## Related Information

[Data Creation and Modification in UltraLite.NET Using SQL Statements \[page 604\]](#)

## 2.5.1 Row Navigation in UltraLite.NET

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

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

### **MoveAfterLast**

moves to a position after the last row.

### **MoveBeforeFirst**

moves to a position before the first row.

### **MoveFirst**

moves to the first row.

### **MoveLast**

moves to the last row.

### **MoveNext**

moves to the next row.

### **MovePrevious**

moves to the previous row.

### **MoveRelative(offset)**

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

## Example

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

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

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

## Example

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

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

## 2.5.2 UltraLite Modes

An UltraLite mode determines the purpose for which the values in the buffer are 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.

## 2.5.3 Row Insertion in UltraLite.NET

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

The order of row insertion into the table has no significance.

## Example

The following code inserts a new row.

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

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

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

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

## 2.5.4 Row Updates

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.

### ⚠ Caution

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

#### In this section:

[Updating a Row in a Table Using UltraLite.NET \[page 613\]](#)

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

## Related Information

[Transaction Management in UltraLite.NET \[page 618\]](#)

### 2.5.4.1 Updating a Row in a Table Using UltraLite.NET

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

#### Procedure

1. Move to the row you want to update.

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

2. Enter update mode.

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

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

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

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

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

4. Execute the Update.

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

## Results

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

## 2.5.5 Row Searches

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

### i 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.

#### In this section:

[Searching for a Row with the Find and Lookup Methods \[page 615\]](#)

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

## 2.5.5.1 Searching for a Row with the Find and Lookup Methods

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

### Procedure

1. Enter find or lookup mode.

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

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

2. Set the search values.

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

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

3. Search for the row.

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

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

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

4. Search for the next instance of the row.

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

### Results

The cursor points to the desired row.

### Next Steps

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

## 2.5.6 Row Retrieval

Row operations can be managed using various methods in the ULTable class.

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.

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

The following code retrieves the value of the cust\_id column, which is an integer.

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

### Modifying Column Values

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

#### Example

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

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

By assigning values to these properties you do not alter the value of the data in the database. You can assign values to the properties even if you are before the first row or after the last row of the table, but it is an error to



try to access data when the current row is at one of these positions, for example, by assigning the property to a variable.

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

## Casting Values

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

## Related Information

[CAST Function \[Data Type Conversion\] \[page 358\]](#)

[CONVERT Function \[Data Type Conversion\] \[page 367\]](#)

## 2.5.7 Row Deletions in UltraLite.NET

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

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

## Example

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

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

## 2.6 Transaction Management in UltraLite.NET

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

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

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

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

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

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

### i Note

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

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

## 2.7 Schema Information in UltraLite.NET

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

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

You can access the following schema objects and information:

### ULDatabaseSchema

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

Call `ULConnection.Schema` to obtain a `ULDatabaseSchema` object.

#### **ULTableSchema**

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

Call `ULTable.Schema` to obtain a `ULTableSchema` object.

#### **ULIndexSchema**

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

Call the `ULTableSchema.GetIndex`, `ULTableSchema.GetOptimalIndex`, or `ULTableSchema.GetPrimaryKey` method to obtain a `ULIndexSchema` object.

## **2.8 Error Handling in UltraLite.NET**

You can use the standard .NET error-handling features to handle errors. Most UltraLite methods throw `ULException` errors.

You can use `ULException.NativeError` to retrieve the `ULSQLCode` value assigned to this error. `ULException` has a `Message` property, which you can use to obtain a descriptive text of the error. `ULSQLCode` errors are negative numbers indicating the error type.

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

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

## 2.9 MobiLink Data Synchronization in UltraLite.NET

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

You can 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 17 installation.

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

### In this section:

#### [Synchronization Initiation in a C# Application \[page 620\]](#)

Use the SyncParms object to initiate synchronization in a C# application.

#### [Microsoft ActiveSync Synchronization Setup in UltraLite.NET \[page 621\]](#)

Microsoft ActiveSync synchronization can be added to an UltraLite.NET application, and your application can be registered for use with Microsoft ActiveSync on your end users' computers.

## Related Information

[UltraLite Clients \[page 73\]](#)

### 2.9.1 Synchronization Initiation in a C# Application

Use the SyncParms object to initiate synchronization in a C# application.

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

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

## 2.9.2 Microsoft ActiveSync Synchronization Setup in UltraLite.NET

Microsoft ActiveSync synchronization can be added to an UltraLite.NET application, and your application can be registered for use with Microsoft ActiveSync on your end users' computers.

Microsoft ActiveSync synchronization can only be initiated by Microsoft ActiveSync. Microsoft ActiveSync initiates synchronization when the device is placed in the cradle or when [Synchronize](#) is selected from the Microsoft ActiveSync window.

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

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

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

If your application is multithreaded, use a separate connection and use the `lock` keyword in Microsoft C# or `SyncLock` keyword in Microsoft 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.

## 2.10 How to Deploy UltraLite.NET Applications

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

### In this section:

#### [Deploying an UltraLite.NET Application for Microsoft Windows Mobile \[page 622\]](#)

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

#### [Deploying an UltraLite.NET Application for Windows Mobile \(UltraLite Engine\) \[page 623\]](#)

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

## Related Information

[UltraLite Application Build and Deployment Specifications \[page 124\]](#)

### 2.10.1 Deploying an UltraLite.NET Application for Microsoft Windows Mobile

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

#### Procedure

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

Synchronization type	Parameter settings
TCP/IP	Set the <code>Stream</code> synchronization parameter to <code>tcpip</code> .
HTTP	Set the <code>Stream</code> synchronization parameter to <code>http</code> .
RSA TLS	Set the <code>Stream</code> synchronization parameter to <code>tls</code> .
RSA HTTPS	Set the <code>Stream</code> synchronization parameter to <code>https</code> .

3. When using RSA end-to-end encryption, set the protocol option `e2ee_public_key= key-file`.
4. When using ZLIB compression, set the protocol option `compression=zlib`.
5. Add references to:
  - Sap.Data.UltraLite
  - Sap.Data.UltraLite.resources
6. Deploy the following files:
  - `%SQLANY17%\UltraLite\UltraLite.NET\Assembly\V2\Sap.Data.UltraLite.dll`.
  - `%SQLANY17%\UltraLite\UltraLite.NET\Assembly\V2\en\Sap.Data.UltraLite.resources.dll`.
  - `ulnet17.dll`, located in `%SQLANY17%\UltraLite\UltraLite.NET\CE\Arm50` for Microsoft Windows Mobile. For Microsoft Windows, it is located in `%SQLANY17%\UltraLite\UltraLite.NET\x64` or `%SQLANY17%\UltraLite\UltraLite.NET\win32`.
7. Deploy the files appropriate for your application:

- When using ZLIB compression, `m1czlib17.dll`.
- When using RSA TLS, RSA HTTPS, or RSA E2EE, `m1crsa17.dll`.

For Microsoft Windows Mobile, the files are located in `%SQLANY17%\UltraLite\CE\Arm.50`. For Microsoft Windows, the files are located in `%SQLANY17%\UltraLite\Windows\x64` or `%SQLANY17%\UltraLite\Windows\x86`.

## Results

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

## Next Steps

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

## Related Information

[UltraLite Database Deployment Techniques \[page 130\]](#)

[UltraLite Application Build and Deployment Specifications \[page 124\]](#)

## 2.10.2 Deploying an UltraLite.NET Application for Windows Mobile (UltraLite Engine)

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

### Procedure

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

Synchronization type	Parameter settings
TCP/IP	Set the <i>Stream</i> synchronization parameter to tcpip.
HTTP	Set the <i>Stream</i> synchronization parameter to http.
RSA TLS	Set the <i>Stream</i> synchronization parameter to tls.
RSA HTTPS	Set the <i>Stream</i> synchronization parameter to https.

- When using RSA end-to-end encryption, set the protocol option `e2ee_public_key= key-file`.
- When using ZLIB compression, set the protocol option `compression=zlib`.
- Add references to:
  - Sap.Data.UltraLite
  - Sap.Data.UltraLite.resources
- Deploy the following files:
  - `%SQLANY17%\UltraLite\UltraLite.NET\Assembly\V2\Sap.Data.UltraLite.dll`.
  - `%SQLANY17%\UltraLite\UltraLite.NET\Assembly\V2\en\Sap.Data.UltraLite.resources.dll`.
  - `ulnetclient17.dll`, located in `%SQLANY17%\UltraLite\UltraLite.NET\CE\Arm50` for Windows Mobile. For Windows, it is located in `%SQLANY17%\UltraLite\UltraLite.NET\x64` or `%SQLANY17%\UltraLite\UltraLite.NET\win32`.
- Deploy the files appropriate for your application:
  - `uleng17.exe`.
  - When using ZLIB compression, `mlczlib17.dll`.
  - When using RSA TLS, RSA HTTPS, or RSA E2EE, `mlcrsa17.dll`.

For Windows Mobile, the files are located in `%SQLANY17%\UltraLite\CE\Arm.50`. For Windows, the files are located in `%SQLANY17%\UltraLite\Windows\x64` or `%SQLANY17%\UltraLite\Windows\x86`.

## Results

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

## Next Steps

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



## Related Information

[UltraLite Database Deployment Techniques \[page 130\]](#)

[UltraLite Application Build and Deployment Specifications \[page 124\]](#)

## 2.11 Tutorial: Building a Microsoft Windows Mobile Application Using UltraLite.NET

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

### Prerequisites

This tutorial assumes the following:

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

### Context

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

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

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

- Ensure Microsoft Visual Studio is not running.
- For Microsoft Visual Studio 2005 or later, run `installULNet.exe` from the folder named `%SQLANY17%\UltraLite\UltraLite.NET\Assembly\v2\`. This task may require administrator privileges.

1. [Lesson 1: Creating a Microsoft Visual Studio Project \[page 626\]](#)  
In this lesson, you create and configure a new Microsoft Visual Studio application. You can choose whether to use Microsoft Visual Basic or C# as your programming language.
2. [Lesson 2: Creating an UltraLite Database \[page 629\]](#)  
In this lesson, you create an UltraLite database using SQL Central on a desktop PC.
3. [Lesson 3: Adding Database Connection Controls to the Application \[page 631\]](#)  
In this lesson, you add a control to your UltraLite.NET application that establishes a connection to an UltraLite database.
4. [Lesson 4: Inserting, Updating, and Deleting Data \[page 633\]](#)  
In this lesson, you add code to your application that uses Dynamic SQL to modify the data in your database.
5. [Lesson 5: Building and Deploying the Application \[page 637\]](#)  
In this lesson, you build your application and deploy it to a remote device or emulator.
6. [Code Listing for C# Tutorial \[page 639\]](#)  
Following is the complete code for the tutorial program described in the preceding sections.
7. [Code Listing for Microsoft Visual Basic Tutorial \[page 641\]](#)  
Following is the complete code for the tutorial program described in the preceding sections.

## Related Information

[Creating an UltraLite Database with the Create Database Wizard \[page 28\]](#)

### 2.11.1 Lesson 1: Creating a Microsoft Visual Studio Project

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

#### Prerequisites

This lesson assumes that you have installed the required software.

#### Context

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

## Procedure

1. Create a Microsoft Visual Studio project.
  - In the Microsoft Visual Studio *File* menu, click **► New ► Project ►**.
  - The *New Project* window appears. In the left pane, expand either the *Visual Basic* folder or the *Visual C#* folder. Click *Smart Device* for the project type.  
In the right pane, click a *Smart Device Project* and name your project VApp or CSApp, depending on whether you are using Microsoft Visual Basic or C# for the programming language.
  - Enter a Location of `C:\tutorial\uldotnet` and click *OK*.
  - Click *Windows Mobile 5.0 Pocket PC SDK* as the target platform and *.NET Compact Framework Version 3.5* as the target Microsoft .NET Compact Framework version. Click *OK*.
2. Add references to your project.
  - Add the Sap.Data.UltraLite assembly and the associated resources to your project.
    1. From the *Project* menu, click *Add Reference*.
    2. Click *Sap.Data.UltraLite* and *Sap.Data.UltraLite EN* (for English) in the list of available references. Click *OK* to add them to the list of selected components.  
If your desired language is not English, click *Browse* and locate *Sap.Data.UltraLite xx* in the `UltraLite\UltraLite.NET\ce\Assembly\v2\xx` subdirectory of your SQL Anywhere installation, where xx is a two-letter abbreviation for your desired language (for example, use *en* for English). Click `Sap.Data.UltraLite.resources.dll` and click *Open*.
  - Link the UltraLite component to your project.  
In this step, ensure that you add a link to the component, and that you do not open the component.
    1. From the *Project* menu, click *Add Existing Item* and browse to the `UltraLite\UltraLite.NET\ce` subdirectory of your SQL Anywhere installation.
    2. In the *Objects of Type* list, click *Executable Files*.
    3. Open the folder corresponding to the processor of the Microsoft Windows Mobile device you are using. For Microsoft Visual Studio 2005 and later, open the `arm.50` folder. Click `ulnet17.dll`; Click the arrow on the *Add* button and click *Add as Link*.
3. Create a form for your application.

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

Type	Design - name
Button	btnInsert
Button	btnUpdate
Button	btnDelete
TextBox	txtName
ListBox	lbNames
Label	laName

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 Microsoft Visual Studio project properly.

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

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

- b. From the *Debug* menu, click *Start Debugging*.

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

The deployment may take some time.

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

## Results

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

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building a Microsoft Windows Mobile Application Using UltraLite.NET \[page 625\]](#)

**Next task:** [Lesson 2: Creating an UltraLite Database \[page 629\]](#)

## 2.11.2 Lesson 2: Creating an UltraLite Database

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

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

### Procedure

1. Click **Start** > **Programs** > **SQL Anywhere 17** > **Administration Tools** > **SQL Central**.
2. Use the UltraLite plug-in for SQL Central to create a database in the same directory as your application.

From the **Tools** menu, click **UltraLite 17** > **Create Database**.

In general, the default database characteristics provided by SQL Central are suitable. Note the following characteristics:

#### Database file name

c:\tutorial\uldotnet\VBApp\VBApp.udb or c:\tutorial\uldotnet\CSApp\CSApp.udb, depending on your application type.

#### DBA user ID and password

Set to DBA and sql, respectively, for the purposes of examples in this documentation.

#### Collation sequence

Use the default collation.

### Use case-sensitive string comparisons

This option should not be on.

Click *Finish* and connect to the UltraLite database.

3. Create a new UltraLite table by highlighting the *Tables* folder icon in the SQL Central tree view and then click **File > New > Table**. Note the following characteristics:

#### Table name

Type **Names**.

#### Columns

Create columns in the **Names** table with the following attributes:

Column Name	Data Type (Size)	Nulls	Unique	Default value
ID	Integer	No	Yes (primary key)	Global autoincrement
Name	Varchar(30)	No	No	None

#### Primary key

Specify the ID column as primary key.

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

## Results

An UltraLite database is created.

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building a Microsoft Windows Mobile Application Using UltraLite.NET \[page 625\]](#)

**Previous task:** [Lesson 1: Creating a Microsoft Visual Studio Project \[page 626\]](#)

**Next task:** [Lesson 3: Adding Database Connection Controls to the Application \[page 631\]](#)

## Related Information

[Creating an UltraLite Database with the Create Database Wizard \[page 28\]](#)

### 2.11.3 Lesson 3: Adding Database Connection Controls to the Application

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

#### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

#### Procedure

1. Double-click the form to open the source file (`Form1.cs` or `Form1.vb`).
2. Add code to import the `Sap.Data.UltraLite` namespace.

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

```
//Microsoft Visual C#  
using Sap.Data.UltraLite;
```

```
'Microsoft Visual Basic  
Imports Sap.Data.UltraLite
```

3. Add global variables to the form declaration.

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

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

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

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

These variables are used as follows:

### **ULConnection**

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

### **ids**

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

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

4. Double-click a blank area of your form to create a Form1\_Load method.

This method performs the following tasks:

- Open a connection to the database using the connection parameters set in the ulConnectionParms1 control.
- Call the RefreshListBox method (defined later in this tutorial).
- Print (display) and error message if an error occurs. For SQL Anywhere errors, the code also prints the error code.

For C#, add the following code to the Form1\_Load method.

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

For Microsoft Visual Basic, add the following code to the Form1\_Load method.

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

5. Build the project.

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



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

## Results

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

**Task overview:** [Tutorial: Building a Microsoft Windows Mobile Application Using UltraLite.NET \[page 625\]](#)

**Previous task:** [Lesson 2: Creating an UltraLite Database \[page 629\]](#)

**Next task:** [Lesson 4: Inserting, Updating, and Deleting Data \[page 633\]](#)

## 2.11.4 Lesson 4: Inserting, Updating, and Deleting Data

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

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

### Context

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

### Procedure

1. Right-click the form and click [View Code](#).
2. Add a method of the Form1 class to update and populate the listbox. This method carries out the following tasks:
  - Clears the listbox.
  - Instantiates a ULCommand object and assigns it a SELECT query that returns data from the Names table in the database.

- Executes the query, returning a result set as a ULDataReader.
- Instantiates an integer array with length equal to the number of rows in the result set.
- Populates the listbox with the names returned in the ULDataReader and populates the integer array with the ids returned in the ULDataReader.
- Closes the ULDataReader.
- If an error occurs, prints the error message. For SQL errors, the code also prints the error code.

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

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

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

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

### 3. Build the project.

Building the project should result in no errors.

4. On the form design tab, double-click *Insert* to create a `btnInsert_Click` method. This method carries out the following tasks:
  - Instantiates a `ULCommand` object and assigns it an `INSERT` statement that inserts the value in the text box into the database.
  - Executes the statement.
  - Disposes of the `ULCommand` object.
  - Refreshes the listbox.
  - If an error occurs, prints the error message. For SQL errors, the code also prints the error code.

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

```
//Microsoft 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 Microsoft Visual Basic, add the following code to the `btnInsert_Click` method.

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

5. On the form design tab, double-click *Update* to create a `btnUpdate_Click` method. This method carries out the following tasks:
  - Instantiates a `ULCommand` object and assigns it an `UPDATE` statement that inserts the value in the text box into the database based on the associated ID.
  - Executes the statement.
  - Disposes of the `ULCommand` object.
  - Refreshes the listbox.
  - If an error occurs, prints the error message. For SQL errors, the code also prints the error code.

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

```
//Microsoft 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 Microsoft Visual Basic, add the following code to the btnUpdate\_Click method.

```

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

```

6. On the form design tab, double-click *Delete* to create a btnDelete\_Click method. Add code to perform the following tasks:
- Instantiates a ULCommand object and assigns it a DELETE statement. The DELETE statement deletes the selected row from the database, based on the associated ID from the integer array ids.
  - Executes the statement.
  - Disposes of the ULCommand object.
  - Refreshes the listbox.
  - If an error occurs, displays the error message. For SQL errors, the code also displays the error code.

For Microsoft C#, add the following code to the btnDelete\_Click method.

```

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

```

For Microsoft Visual Basic, add the following code to the btnDelete\_Click method.

```

'Microsoft Visual Basic
Try
    Dim RowsDeleted As Long

```

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

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

## Results

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

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building a Microsoft Windows Mobile Application Using UltraLite.NET \[page 625\]](#)

**Previous task:** [Lesson 3: Adding Database Connection Controls to the Application \[page 631\]](#)

**Next task:** [Lesson 5: Building and Deploying the Application \[page 637\]](#)

## Related Information

[Data creation and modification in UltraLite.NET using the ULTable Class \[page 610\]](#)

## 2.11.5 Lesson 5: Building and Deploying the Application

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

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

## Procedure

1. Build the solution.

Ensure that your application builds without errors.

2. Choose the deployment target.

The deployment target must match the version of `ulnet17.dll` that you included in your application.

3. Click **Debug > Start**.

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

4. If errors are reported, use the following checklist to ensure that your deployment was completed successfully:

- Confirm that the application is deployed into `\Program Files\appname`, where `appname` is the name you gave your application in Lesson 1 (CSApp or VBApp).
- Confirm that the path to the database file in your application code is correct.
- Confirm that you chose Link File when adding the database file to the project and you set the Build Action to Content Only and Copy to Output Directory is set to Copy Always. If you did not set these options correctly, the files will not be deployed to the device.
- Ensure that you added a reference to the correct version of `ulnet17.dll` for your target platform, or ran the Microsoft Windows Mobile installer. For versions of Microsoft Windows Mobile earlier than Microsoft Windows Mobile 5.0, if you switch between the emulator and a real device, you must change the version of the library that you use.
- You may want to exit the emulator without saving the emulator state. Redeploying the application copies all required files to the emulator, and ensures there are no version problems.

5. Test your application:

- a. Insert data into the database.

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

- b. Update data in the database.

Click a name in the listbox. Enter a new name in the text box. Click *Update*.

The new name should now appear in place of the old name in the listbox.

- c. Delete data from the database.

Click a name in the list. Click *Delete*.

The name no longer appears in the list.

## Results

The application is tested and can be deployed.

**Task overview:** [Tutorial: Building a Microsoft Windows Mobile Application Using UltraLite.NET \[page 625\]](#)

Previous task: [Lesson 4: Inserting, Updating, and Deleting Data \[page 633\]](#)

Next: [Code Listing for C# Tutorial \[page 639\]](#)

## 2.11.6 Code Listing for C# Tutorial

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

```
using Sap.Data.UltraLite;
using System;
using System.Linq;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace CSApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private ULConnection Conn;
        private int[] ids;
        private void Form1_Load(object sender, EventArgs e)
        {
            try
            {
                String ConnString = "dbf=\\Program Files\\CSApp\\CSApp.udb";
                Conn = new ULConnection(ConnString);
                Conn.Open();
                Conn.DatabaseID = 1;
                RefreshListBox();
            }
            catch (System.Exception t)
            {
                MessageBox.Show("Exception: " + t.Message);
            }
        }
        private void RefreshListBox()
        {
            try
            {
                long NumRows;
                int I = 0;
                lbNames.Items.Clear();
                using (ULCommand cmd = Conn.CreateCommand())
                {
                    cmd.CommandText = "SELECT ID, Name FROM Names";
                    using (ULDataReader dr = cmd.ExecuteReader())
                    {
                        dr.MoveBeforeFirst();
                        NumRows = dr.RowCount;
                        ids = new int[NumRows];
                        while (dr.MoveNext())
                        {
                            lbNames.Items.Add(
                                dr.GetString(1));
                            ids[I] = dr.GetInt32(0);
                        }
                    }
                }
            }
        }
    }
}
```

```

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

```



```

        catch (Exception err)
        {
            MessageBox.Show("Exception: " + err.Message);
        }
    }
}

```

**Parent topic:** [Tutorial: Building a Microsoft Windows Mobile Application Using UltraLite.NET \[page 625\]](#)

**Previous task:** [Lesson 5: Building and Deploying the Application \[page 637\]](#)

**Next:** [Code Listing for Microsoft Visual Basic Tutorial \[page 641\]](#)

## 2.11.7 Code Listing for Microsoft Visual Basic Tutorial

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

```

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

```

```

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

```

**Parent topic:** [Tutorial: Building a Microsoft Windows Mobile Application Using UltraLite.NET \[page 625\]](#)

**Previous:** [Code Listing for C# Tutorial \[page 639\]](#)

## 3 UltraLite - C++ Programming

This book describes UltraLite C++ programming interface. With UltraLite, you can develop and deploy database applications to handheld, or mobile devices, including Apple iPhone and iPad, Google Android, and embedded devices.

### In this section:

#### [System Requirements and Supported Platforms \[page 643\]](#)

UltraLite C++ supports Microsoft Windows Mobile, Linux, Apple iOS, and various other platforms. Third-party software is required for UltraLite database development.

#### [UltraLite Application Development Using C++ \[page 644\]](#)

The C++ interfaces provide the following benefits for UltraLite developers:

#### [Tutorial: Building a Windows Application using the C++ API \[page 717\]](#)

This tutorial guides you through the process of building an UltraLite C++ application. The application is built for Windows desktop operating systems, and runs at a command prompt.

#### [API Reference \[page 728\]](#)

Use the UltraLite C++ API to develop mobile applications.

### 3.1 System Requirements and Supported Platforms

UltraLite C++ supports Microsoft Windows Mobile, Linux, Apple iOS, and various other platforms. Third-party software is required for UltraLite database development.

#### Development Platforms

To develop applications using UltraLite C++, you must have the following:

- A Microsoft Windows, Linux, or Mac desktop as a development platform.
- A supported Microsoft or GNU C++ compiler.

#### Target Platforms

UltraLite C++ supports the following target platforms:

- Apple iOS
- Apple macOS

- Linux
- Embedded Linux
- Microsoft Windows
- Microsoft Windows Mobile

For the most up-to-date platform version information, go to <http://scn.sap.com/docs/DOC-35654#UL>.

## 3.2 UltraLite Application Development Using C++

The C++ interfaces provide the following benefits for UltraLite developers:

- A small, high-performance database store with native synchronization.
- The power, efficiency, and flexibility of the C++ language.
- The ability to deploy applications on Microsoft Windows Mobile, Microsoft Windows desktop platforms, Linux, and Apple iOS.

All UltraLite C++ interfaces utilize the same UltraLite runtime engine. The APIs each provide access to the same underlying functionality.

### In this section:

#### [UltraLite C++ Application Development \[page 644\]](#)

UltraLite C++ provides database functionality and synchronization to Microsoft Windows Mobile devices, Microsoft Windows desktop platforms, Linux desktop, embedded Linux, Apple macOS desktop platforms, and Apple iOS devices.

#### [UltraLite C++ Application Development Using Embedded SQL \[page 676\]](#)

You can write database access codes for Embedded SQL UltraLite applications.

#### [UltraLite Application Development for Microsoft Windows Mobile \[page 708\]](#)

Microsoft Visual Studio 2005 and later can be used to develop applications for the Microsoft Windows Mobile environment.

### 3.2.1 UltraLite C++ Application Development

UltraLite C++ provides database functionality and synchronization to Microsoft Windows Mobile devices, Microsoft Windows desktop platforms, Linux desktop, embedded Linux, Apple macOS desktop platforms, and Apple iOS devices.

### In this section:

#### [Quick Start Guide to UltraLite C++ Application Development \[page 645\]](#)

UltraLite C++ development involves use of a `ULDatabaseManager` object in your application.

#### [Apple iOS and macOS Considerations \[page 646\]](#)

Several design and development decisions need to be made when developing for Apple iOS or macOS platforms.

#### [UltraLite Database Connections \[page 647\]](#)

UltraLite applications must connect to the database before performing operations on its data.

#### [Data Creation and Modification in UltraLite C++ Using SQL Statements \[page 649\]](#)

UltraLite applications can access table data by executing SQL statements or using the ULTable class.

#### [Data Creation and Modification in UltraLite C++ Using the ULTable Class \[page 657\]](#)

UltraLite applications can access table data by executing SQL statements or using the ULTable class.

#### [Transaction Management in UltraLite C++ \[page 664\]](#)

Transactions are started implicitly by the first statement to modify the database, and must be explicitly committed or rolled back.

#### [Schema Information in UltraLite C++ \[page 664\]](#)

You can programmatically retrieve result set or database structure descriptions. These descriptions are known as schema information, and this information is available through the UltraLite C API schema classes.

#### [Error Handling \[page 665\]](#)

The UltraLite C++ API includes a ULError object that should be used to retrieve error information. Several methods in the API return a boolean value, indicating whether the method call was successful. In some instances, null is returned when an error occurs.

#### [MobiLink Data Synchronization in UltraLite C++ \[page 666\]](#)

UltraLite applications can synchronize data with a central database. Synchronization requires the MobiLink synchronization software included with SQL Anywhere.

#### [Closing the UltraLite Database Connection \[page 666\]](#)

Release software resources when they are no longer being used to prevent the UltraLite database file from remaining in use for as long as the application has a connection to the database.

#### [How to Build and Deploy UltraLite C++ Applications \[page 667\]](#)

When building a C++ application that does not use the UltraLite engine, you can either link to a static UltraLite runtime library or, on Windows and Windows Mobile, you can link to an import library and load the UltraLite runtime code dynamically when the application starts.

### **3.2.1.1 Quick Start Guide to UltraLite C++ Application Development**

UltraLite C++ development involves use of a ULDatabaseManager object in your application.

The following procedure is generally used when creating an application using the UltraLite C++ API:

1. Initialize a ULDatabaseManager object.
2. (Optional) Enable features in the UltraLite runtime library.
3. Use an UltraLite database. You can open a connection to an existing database, create a new one, drop an existing database, or validate that an existing database has no file corruption.
4. Finalize the ULDatabaseManager object.

The ULDatabaseManager object should only be initialized once in your application and then finalized when your application is terminating. All methods on the ULDatabaseManager class are static. Use the ULError class to get error information throughout your UltraLite application.

## Related Information

[How to Build and Deploy UltraLite C++ Applications \[page 667\]](#)

### 3.2.1.2 Apple iOS and macOS Considerations

Several design and development decisions need to be made when developing for Apple iOS or macOS platforms.

#### Development Environment

The development environment for Apple iOS and macOS is Xcode.

#### Build Settings

To reference the UltraLite header files and library it is convenient to create a user-defined build setting set to the location of the SQL Anywhere installation directory. For example, set `SQLANY_ROOT` to `/Applications/SQLAnywhere17`. To create this setting, open the project editor's *Build* pane and click *Add User-Defined Setting* and enter the name and value.

#### Include Files

To find the UltraLite include files, add `$(SQLANY_ROOT)/sdk/include` to the *User Header Search Paths* (`USER_HEADER_SEARCH_PATHS`) build setting.

#### Unsupported MobiLink Client Network Protocol Options

UltraLite for Apple iOS and/or macOS does not support the following MobiLink client network protocol options:

- `certificate_company`
- `certificate_unit`
- `client_port`
- `identity`
- `identity_password`
- `network_leave_open`
- `network_name`

## Encryption

FIPS-certified encryption standards are not supported.

To use end-to-end encryption when synchronizing Apple iOS and macOS UltraLite clients with a MobiLink server, you must encapsulate your public keys in a the PEM encoded X509 certificate (as opposed to a PEM public key file). To create a PEM encoded X509 certificate with an E2EE private key use the certificate creation utility, `createcert`.

When developing UltraLite applications for the iPhone, you must include the certificate in the Resources folder in your Xcode project. UltraLite synchronization logic searches for the certificate file in the Main Resource Bundle (`mainBundle`) of the iPhone development package if the `trusted_certificates` or `e2ee_public_key` options are assigned.

## Debugging iPhone Applications

The Xcode debugger (GDB) has support for stepping through and breaking on `longjmp()` calls. Applications typically do not use `longjmp`, but the UltraLite runtime library does internally (sometimes, when an error is signaled, for instance). This may cause problems when tracing through application code and stepping over UltraLite calls. If you step over an UltraLite call and get an error from the debugger: Restart the program, set a breakpoint after the problematic line and, instead of stepping over the problematic line, use the *Continue* command - this will have the same effect because the debugger will stop at the following breakpoint, but should avoid problems related to `longjmp` calls. The most likely place to encounter this is when using `OpenConnection` to open an existing database or determine that the database doesn't exist (an error is signaled when the database doesn't exist).

### 3.2.1.3 UltraLite Database Connections

UltraLite applications must connect to the database before performing operations on its data.

The `ULDatabaseManager` class is used to open a connection to a database. The `ULDatabaseManager` class returns a non-null `ULConnection` object when a connection is established. Use the `ULConnection` object to perform the following tasks:

- Commit or roll back transactions.
- Synchronize data with a MobiLink server.
- Access tables in the database.
- Work with SQL statements.
- Handle errors in your application.

Ensure you specify a writable path for the database file. Use the `NSSearchPathForDirectoriesInDomains` method to query the `NSDocumentDirectory`, for example.

#### i Note

You can find sample code in the `%SQLANYAMP17%\UltraLite\CustDB\` directory.

## Multithreaded Applications

Each connection and all objects created from it should be used by a single thread. If an application requires multiple threads accessing the UltraLite database, each thread requires a separate connection.

In this section:

[Connecting to an UltraLite Database Using UltraLite C++ \[page 648\]](#)

Use the `ULDatabaseManager` object to create or connect to an UltraLite database named `sample.udb`.

### 3.2.1.3.1 Connecting to an UltraLite Database Using UltraLite C++

Use the `ULDatabaseManager` object to create or connect to an UltraLite database named `sample.udb`.

#### Procedure

1. Initialize the `ULDatabaseManager` object and enable features in UltraLite using the following code:

```
if( !ULDatabaseManager::Init() ) {
    return 0;
}
ULDatabaseManager::EnableAesDBEncryption();

// Use ULDatabaseManager.Fini() when terminating the app.
```

2. Open a connection to an existing database or create a new database if the specified database file does not exist using the following code:

```
ULConnection * conn;
ULError ulerr;

conn = ULDatabaseManager::OpenConnection( "dbf=sample.udb;dbkey=aBcD1234",
&ulerr );
if( conn == NULL ) {
    if( ulerr.GetSQLCode() == SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
        conn =
ULDatabaseManager::CreateDatabase( "dbf=sample.udb;dbkey=aBcD1234", &ulerr );
        if( conn == NULL ) {
            // write code that uses ulerr to determine what happened
            return 0;
        }
        // add code to create the schema for your database
    } else {
        // write code that uses ulerr to determine what happened
        return 0;
    }
}
assert( conn != NULL );
```



In this step, you declare a `ULError` object that contains error information in case the connection is not successful.

## Results

A connection to the `sample.udb` database is established.

### 3.2.1.4 Data Creation and Modification in UltraLite C++ Using SQL Statements

UltraLite applications can access table data by executing SQL statements or using the `ULTable` class.

The following tasks can be performed using SQL statements:

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

#### In this section:

##### [Data Modification in UltraLite C++ Using INSERT, UPDATE, and DELETE \[page 650\]](#)

With UltraLite, you can perform SQL data manipulation by using the `ExecuteStatement` method, a member of the `ULPreparedStatement` class.

##### [Retrieving Data in UltraLite C++ Using SELECT \[page 655\]](#)

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

##### [Schema Description Creation and Retrieval \[page 656\]](#)

The `GetResultSetSchema` method allows you to retrieve schema information about a result set, such as column names, total number of columns, column scales, column sizes, and column SQL types.

##### [SQL Result Set Navigation in UltraLite C++ \[page 656\]](#)

You can navigate through a result set using methods associated with the `ULResultSet` class.

## Related Information

[Data Creation and Modification in UltraLite C++ Using the ULTable Class \[page 657\]](#)

## 3.2.1.4.1 Data Modification in UltraLite C++ Using INSERT, UPDATE, and DELETE

With UltraLite, you can perform SQL data manipulation by using the `ExecuteStatement` method, a member of the `ULPreparedStatement` class.

### In this section:

#### [Inserting a Row in a Table Using UltraLite C++ \[page 650\]](#)

UltraLite indicates query parameters using the `?` character. For any INSERT, UPDATE, or DELETE statement, each `?` is referenced according to its ordinal position in the prepared statement. For example, the first `?` is referred to as parameter 1, and the second as parameter 2.

#### [Deleting a Row in a Table in UltraLite C++ \[page 652\]](#)

UltraLite indicates query parameters using the `?` character. For any INSERT, UPDATE, or DELETE statement, each `?` is referenced according to its ordinal position in the prepared statement. For example, the first `?` is referred to as parameter 1, and the second as parameter 2.

#### [Updating a Row in a Table \[page 653\]](#)

UltraLite indicates query parameters using the `?` character. For any INSERT, UPDATE, or DELETE statement, each `?` is referenced according to its ordinal position in the prepared statement. For example, the first `?` is referred to as parameter 1, and the second as parameter 2.

### 3.2.1.4.1.1 Inserting a Row in a Table Using UltraLite C++

UltraLite indicates query parameters using the `?` character. For any INSERT, UPDATE, or DELETE statement, each `?` is referenced according to its ordinal position in the prepared statement. For example, the first `?` is referred to as parameter 1, and the second as parameter 2.

## Procedure

1. Declare a `ULPreparedStatement` using the following code:

```
ULPreparedStatement * prepStmt;
```

2. Prepare a SQL statement for execution.

The following code prepares an INSERT statement for execution:

```
prepStmt = conn->PrepareStatement("INSERT INTO MyTable(MyColumn1) VALUES  
(?)");
```

3. Check for errors when preparing the statement.

For example, the following code is useful when checking for SQL syntax errors:

```
if( prepStmt == NULL ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error
```

```
    return;  
}
```

4. Set values to replace ? characters in the prepared statement.

The following code sets ? characters to "some value" while error checking. For example, an error is caught when the parameter ordinal is out of range for the number of parameters in the prepared statement.

```
if( !prepStmt->SetParameterString(1, "some value") ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error  
    return;  
}
```

5. Execute the prepared statement, inserting the data into the database.

The following code checks for errors that could occur after executing the statement. For example, an error is returned if a duplicate index value is found in a unique index.

```
bool success;  
success = prepStmt->ExecuteStatement();  
if( !success ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error  
} else {  
    // Use the following line if you are interested in the number of rows  
    inserted ...  
    ul_u_long rowsInserted = prepStmt->GetRowsAffectedCount();  
}
```

6. Clean up the prepared statement resources.

The following code releases the resources used by the prepared statement object. This object should no longer be accessed after the Close method is called.

```
prepStmt->Close();
```

7. Commit the data to the database.

The following code saves the data to the database and prevents data loss. The data from step 5 is lost if the device application terminates unexpectedly before the application completes a commit call.

```
conn->Commit();
```

## Results

A new row is added to MyTable where the MyColumn1 value is set to a string "some value".

## 3.2.1.4.1.2 Deleting a Row in a Table in UltraLite C++

UltraLite indicates query parameters using the ? character. For any INSERT, UPDATE, or DELETE statement, each ? is referenced according to its ordinal position in the prepared statement. For example, the first ? is referred to as parameter 1, and the second as parameter 2.

### Procedure

1. Declare a `ULPreparedStatement` using the following code:

```
ULPreparedStatement * prepStmt;
```

2. Prepare a SQL statement for execution.

The following code prepares a DELETE statement for execution:

```
prepStmt = conn->PrepareStatement("DELETE FROM MyTable(MyColumn1) VALUES  
(?)");
```

3. Check for errors when preparing the statement.

For example, the following code is useful when checking for SQL syntax errors:

```
if( prepStmt == NULL ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error  
    return;  
}
```

4. Set values to replace ? characters in the prepared statement.

The following code sets ? characters to 7 while error checking. For example, an error is caught when the parameter ordinal is out of range for the number of parameters in the prepared statement.

```
ul_s_long value_to_delete = 7;  
if( !prepStmt->SetParameterInt(1, value_to_delete) ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error.  
    return;  
}
```

5. Execute the prepared statement, deleting the data from the database.

The following code checks for errors that could occur after executing the statement. For example, an error is returned if you try deleting a row that has a foreign key referenced to it.

```
bool success;  
success = prepStmt->ExecuteStatement();  
if( !success ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error  
} else {  
    // Use the following line if you are interested in the number of rows  
    deleted ...  
    ul_u_long rowsDeleted = prepStmt->GetRowsAffectedCount();  
}
```

```
}
```

6. Clean up the prepared statement resources.

The following code releases the resources used by the prepared statement object. This object should no longer be accessed after the Close method is called.

```
prepStmt->Close();
```

7. Commit the data to the database.

The following code saves the data to the database and prevents data loss. The data from step 5 is lost if the device application terminates unexpectedly before the application completes a commit call.

```
conn->Commit();
```

## Results

Row entries from MyTable are deleted where the MyColumn value in the table is equal to 7.

### 3.2.1.4.1.3 Updating a Row in a Table

UltraLite indicates query parameters using the ? character. For any INSERT, UPDATE, or DELETE statement, each ? is referenced according to its ordinal position in the prepared statement. For example, the first ? is referred to as parameter 1, and the second as parameter 2.

## Procedure

1. Declare a ULPreparedStatement using the following code:

```
ULPreparedStatement * prepStmt;
```

2. Prepare a SQL statement for execution.

The following code prepares an UPDATE statement for execution:

```
prepStmt = conn->PrepareStatement("UPDATE MyTable SET MyColumn = ? WHERE  
MyColumn = ?");
```

3. Check for errors when preparing the statement.

For example, the following code is useful when checking for SQL syntax errors:

```
if( prepStmt == NULL ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error  
    return;  
}
```

4. Set values to replace ? characters in the prepared statement.

The following code sets ? characters to integer values while error checking. For example, an error is caught when the parameter ordinal is out of range for the number of parameters in the prepared statement.

```
bool success;
success = prepStmt->SetParameterInt( 1, 25 );
if( success ) {
    success = prepStmt->SetParameterInt( 2, -1 );
}
if( !success ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
    return;
}
```

5. Execute the prepared statement, updating the data in the database.

The following code checks for errors that could occur after executing the statement. For example, an error is returned if a duplicate index value is found in a unique index.

```
success = prepStmt->ExecuteStatement();
if( !success ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
} else {
    // if you are interested in the number of rows updated ...
    ul_ulong rowsUpdated = prepStmt->GetRowsAffectedCount();
}
```

6. Clean up the prepared statement resources.

The following code releases the resources used by the prepared statement object. This object should no longer be accessed after the Close method is called.

```
prepStmt->Close();
```

7. Commit the data to the database.

The following code saves the data to the database and prevents data loss. The data from step 5 is lost if the device application terminates unexpectedly before the application completes a commit call.

```
conn->Commit();
```

## Results

In this scenario, row entries from MyTable are updated where the MyColumn value is equal to -1. The value is updated to 25.

## 3.2.1.4.2 Retrieving Data in UltraLite C++ Using SELECT

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

### Procedure

1. Declare the required variables using the following code:

```
ULPreparedStatement * prepStmt;  
ULResultSet * resultSet;
```

2. Prepare a SQL statement for execution.

The following code prepares a SELECT statement for execution:

```
prepStmt = conn->PrepareStatement("SELECT MyColumn1 FROM MyTable");
```

3. Check for errors when preparing the statement.

For example, the following code is useful when checking for SQL syntax errors:

```
if( prepStmt == NULL ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error  
    return;  
}
```

4. Execute the SQL and return a result set object that can be used to move the results of the query.

```
resultSet = prepStmt->ExecuteQuery();  
if( resultSet == NULL ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error  
    prepStmt->Close();  
    return;  
}
```

5. Traverse the rows by calling the Next method. Store the result as a string and store them in a buffer.

The Next method moves to the next row of the result set. The ULResultSet object is positioned on a row if the call returns true; otherwise, if the call returns false, all the rows have been traversed.

```
while( resultSet->Next() ) {  
    char buffer[ 100 ];  
    resultSet->GetString( 1, buffer, 100 );  
    printf( "MyColumn = %s\n", buffer );  
}
```

6. Clean up the prepared statement and result set object resources.

The prepared statement object should not be accessed after the Close method is called.

```
resultSet->Close();  
prepStmt->Close();
```

## Results

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

### 3.2.1.4.3 Schema Description Creation and Retrieval

The GetResultSetSchema method allows you to retrieve schema 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 GetResultSetSchema method to display schema information in a command prompt:

```
const char * name;
int column_count;
const ULResultSetSchema & rss = prepStmt->GetResultSetSchema();
int column_count = rss.GetColumnCount();
for( int i = 1; i < column_count; i++ ) {
    name = rss.GetColumnName( i );
    printf( "id = %d, name = %s\n", i, name );
}
```

In this example, required variables are declared and the ULResultSetSchema object is assigned. It is possible to get a ULResultSetSchema object from the result set object itself, but this example demonstrates how the schema is available after the statement is prepared and before the query is executed. The number of rows in the result set are counted, and the name of each column is displayed.

### 3.2.1.4.4 SQL Result Set Navigation in UltraLite C++

You can navigate through a result set using methods associated with the ULResultSet class.

The result set class provides you with the following methods to navigate a result set:

#### **AfterLast**

Position immediately after the last row.

#### **BeforeFirst**

Position immediately before the first row.

#### **First**

Move to the first row.

#### **Last**

Move to the last row.

#### **Next**



Move to the next row.

#### **Previous**

Move to the previous row.

#### **Relative(offset)**

Move a specified number of rows relative to the current row, as specified by the signed offset value. Positive offset values move forward in the result set, relative to the current pointer position in the result set. Negative offset values move backward in the result set. An offset value of zero does not move the current location, but allows you to repopulate the row buffer.

### **3.2.1.5 Data Creation and Modification in UltraLite C++ Using the ULTable Class**

UltraLite applications can access table data by executing SQL statements or using the ULTable class.

The following tasks can be performed using the ULTable class:

- 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.

#### **⚠ Caution**

Do not update the primary key of a row. Delete the row and add a new row instead.

#### **In this section:**

##### [Row Navigation in UltraLite C++ \[page 658\]](#)

The UltraLite C++ API provides you with several methods to navigate a table to perform a wide range of navigation tasks.

##### [UltraLite Modes \[page 659\]](#)

The UltraLite mode determines how values in the buffer are used.

##### [Row Insertions in UltraLite C++ \[page 660\]](#)

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.

##### [Updating Rows \[page 660\]](#)

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

##### [Find and Lookup Modes for Searching Rows \[page 661\]](#)

UltraLite has different modes of operation for working with data. You can use two of these modes, find and lookup, for searching. The ULTable object has methods corresponding to these modes for locating particular rows in a table.

##### [Access to Values in the Current Row \[page 662\]](#)

Use the ULTable object to manage the column values in a row.

##### [Row Deletions in UltraLite C++ \[page 664\]](#)

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.

## Related Information

[Data Creation and Modification in UltraLite C++ Using SQL Statements \[page 649\]](#)

### 3.2.1.5.1 Row Navigation in UltraLite C++

The UltraLite C++ API provides you with several methods to navigate a table to perform a wide range of navigation tasks.

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

#### **AfterLast**

Position immediately after the last row.

#### **BeforeFirst**

Position immediately before the first row.

#### **First**

Move to the first row.

#### **Last**

Move to the last row.

#### **Next**

Move to the next row.

#### **Previous**

Move to the previous row.

#### **Relative(offset)**

Move a specified number of rows relative to the current row, as specified by the signed offset value. Positive offset values move forward in the result set, relative to the current pointer position in the result set. Negative offset values move backward in the result set. An offset value of zero does not move the current location, but allows you to repopulate the row buffer.

## Example

The following example opens the table named MyTable and displays the value of the column named MyColumn for each row:

```
char buffer[ 100 ];
ul_column_num column_id;
ULTable *tbl = conn->OpenTable( "MyTable" );
```

```

if( tbl == NULL ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
    return;
}
column_id = tbl->GetTableSchema().GetColumnID( "MyColumn" );
if( column_id == 0 ) {
    // the column "MyColumn" likely does not exist.  Handle the error.
    tbl->Close();
    return;
}
while( tbl->Next() ) {
    tbl->GetString( column_id, buffer, 100 );
    printf( "%s\n", buffer );
}
tbl->Close();

```

You expose the rows of the table to the application when you open the ULTable 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 example moves to the first row of the MyTable table as ordered by the ix\_col index:

```

ULTable * tbl = conn->OpenTable( "MyTable", "ix_col" );

```

### 3.2.1.5.2 UltraLite Modes

The UltraLite mode determines how values in the buffer are used.

You can set the UltraLite mode to one of the following:

#### Insert mode

Data in the buffer is added to the table as a new row when the insert method is called.

#### Update mode

Data in the buffer replaces the current row when the update method is called.

#### Find mode

Locates a row whose value exactly matches the data in the buffer when one of the find methods is called.

#### Lookup mode

Locates a row whose value matches or is greater than the data in the buffer when one of the lookup methods is called.

The mode is set by calling the corresponding method to set the mode. For example, InsertBegin, UpdateBegin, FindBegin, and so on.

### 3.2.1.5.3 Row Insertions in UltraLite C++

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.

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.

#### Example

The following code demonstrates new row insertion:

```
ULTable * tbl = conn->OpenTable("MyTable");
bool success;
tbl->InsertBegin(); // enter "Insert mode"
tbl->SetInt("id", 3);
tbl->SetString("lname", "Smith");
tbl->SetString("fname", "Mary");
success = tbl->Insert();
conn->Commit();
tbl->Close();
```

In this example, the tbl variable is set to open MyTable. The values for each column are set in the current row buffer; columns can be referenced name or ID. The Insert method causes the temporary row buffer values to be inserted into the database. The results are then committed and displayed. Resources are freed with the Close method.

### 3.2.1.5.4 Updating Rows

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

#### Procedure

1. Move to the row you want to update.

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

2. Enter update mode.

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

```
tbl->UpdateBegin();
```

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

```
tbl->SetInt("id", 3);
```

4. Execute the Update.

```
tbl->Update();
```

#### Caution

When using the Find and Update methods, your pointer may not be in the expected position after updating a column that is involved in the search criteria. In some instances, a SQL statement can be used to update rows.

## Results

After the update operation, the current row is the row that has been updated.

## Next Steps

Perform additional SQL operations on the database and then execute the Commit method to commit changes to the database. executed.

## Related Information

[Transaction Management in UltraLite C++ \[page 664\]](#)

### 3.2.1.5 Find and Lookup Modes for Searching Rows

UltraLite has different modes of operation for working with data. You can use two of these modes, find and lookup, for searching. The ULTable object has methods corresponding to these modes for locating particular rows in a table.

#### Note

The columns you search with Find and Lookup methods must be in the index that is 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 ULTable 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 ULTable object was opened.

## Example

This example uses a table named MyTable that was created using the following SQL statements:

```
CREATE TABLE MyTable( id int primary key, lname char(100), fname char(100) )
CREATE INDEX ix_lname ON MyTable ( lname )
```

The following code displays all the fname column contents where lname column is "Smith":

```
ULTable * tbl = conn->OpenTable( "MyTable", "ix_lname" );
char buffer[ 100 ];
bool found;
tbl->FindBegin(); // enter "Find mode"
tbl->SetString( "lname", "Smith" ); // set pointer row buffer to "Smith"
found = tbl->FindFirst();
while( found ) {
    tbl->GetString( 3, buffer, 100 );
    printf( "%s\n", buffer );
    found = tbl->FindNext();
}
tbl->Close();
```

In this example, the tbl variable is set to open MyTable using the ix\_lname index so that rows are returned in same order as the lname column. ULTable objects use the values in the row buffer when they execute a find. This buffer is specified as "Smith", as defined by the SetString method. The FindFirst method indicates that traversal should begin at the first row that has lname set to "Smith"; the pointer is positioned after the last row of the table if there are no rows where lname is set to "Smith". The fname is retrieved by the GetString method because The fname has a column ID of 3. The results are then displayed, and the resources are freed.

### 3.2.1.5.6 Access to Values in the Current Row

Use the ULTable object to manage the column values in a row.

A ULTable 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 ULTable 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 the columns in that row.

## Retrieving Column Values

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

The following example demonstrates two ways to get an age value out of an open table, assuming that age is the first column of the table:

```
ul_s_long age1 = tbl->GetInt( 1 );
ul_s_long age2 = tbl->GetInt( "age" );
assert( age1 == age2 );
```

Using the column ID version of value retrieval has performance benefits when values are retrieved in a loop.

## Modifying Column Values

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

For example, the following code demonstrates two ways to set a string value for a row with string columns of lname and fname, assuming that lname is the first column in the table.

```
tbl->SetString( 1, last_name );
tbl->SetString( "fname", first_name );
```

By setting column values, you do not directly alter the data in the database. You can assign values to the columns, even if you are before the first row or after the last row of the table. Do not attempt to access data when the current row is undefined. For example, attempting to fetch the column value in the following example is incorrect:

```
// This code is incorrect
tbl->BeforeFirst();
tbl = tbl.GetInt( cust_id );
```

## Casting Values

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

### 3.2.1.5.7 Row Deletions in UltraLite C++

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

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

#### Example

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

```
tbl->First();  
tbl->Delete();
```

### 3.2.1.6 Transaction Management in UltraLite C++

Transactions are started implicitly by the first statement to modify the database, and must be explicitly committed or rolled back.

To commit a transaction, use the `ULConnection.Commit` method.

To rollback a transaction, use the `ULConnection.Rollback` method.

### 3.2.1.7 Schema Information in UltraLite C++

You can programmatically retrieve result set or database structure descriptions. These descriptions are known as schema information, and this information is available through the UltraLite C API schema classes.

#### i Note

You cannot modify the schema using the UltraLite C API. You can only retrieve the schema information.

You can access the following schema objects and information:

#### **ULResultSetSchema**

Describes a query or data in a table. It exposes the identifier, name, and type information of each column, and the number of columns in the table. `ULResultSetSchema` classes can be retrieved from the following classes:

- `ULPreparedStatement`
- `ULResultSet`
- `ULTable`

#### **ULDatabaseSchema**



Exposes the number and names of the tables and publications in the database, and the global properties such as the format of dates and times. `ULDatabaseSchema` classes can be retrieved from `ULConnection` classes.

### **ULTableSchema**

Exposes information about the column and index configurations. The column information in the `ULTableSchema` class complements the information available from the `ULResultSetSchema` class. For example, you can determine whether columns have default values or permit null values. `ULTableSchema` classes can be retrieved from `ULTable` classes.

### **ULIndexSchema**

Returns information about the column in the index. `ULIndexSchema` classes can be retrieved from `ULTableSchema` classes.

The `ULResultSetSchema` class is returned as a constant reference unlike the `ULDatabaseSchema`, `ULTableSchema` and `ULIndexSchema` classes, which are returned as pointers. You cannot close a class that returns a constant reference but you must close classes that are returned as pointers.

The following code demonstrates proper and improper use of schema class closure:

```
// This code demonstrates proper use of the ULResultSetSchema class:
const ULResultSetSchema & rss = prepStmt->GetResultSetSchema();
c_count = prepStmt->GetSchema().GetColumnCount();

// This code demonstrates proper use of the ULDatabaseSchema class:
ULDatabaseSchema * dbs = conn->GetResultSetSchema();
t_count = dbs->GetTableCount();
dbs->Close(); // This line is required.
// This code demonstrates improper use of the ULDatabaseSchema class
// because the object needs to be closed using the Close method:
t_count = conn->GetResultSetSchema()->GetTableCount();
```

## **3.2.1.8 Error Handling**

The UltraLite C++ API includes a `ULError` object that should be used to retrieve error information. Several methods in the API return a boolean value, indicating whether the method call was successful. In some instances, null is returned when an error occurs.

The `ULConnection` object contains a `GetLastError` method, which returns a `ULError` object.

Use `SQLCode` to diagnose an error. In addition to the `SQLCode`, you can use the `GetParameterCount` and `GetParameter` methods to determine whether additional parameters exist to provide additional information about the error.

In addition to explicit error handling, UltraLite supports an error callback function. If you register a callback function, UltraLite calls the function whenever an UltraLite error occurs. The callback function does not control application flow, but does enable you to be notified of all errors. Use of a callback function is particularly helpful during application development and debugging.

## **Related Information**

[Tutorial: Building a Windows Application using the C++ API \[page 717\]](#)

### 3.2.1.9 MobiLink Data Synchronization in UltraLite C++

UltraLite applications can synchronize data with a central database. Synchronization requires the MobiLink synchronization software included with SQL Anywhere.

The UltraLite C++ API supports TCP/IP, TLS, HTTP, and HTTPS synchronization. Synchronization is initiated by the UltraLite application. The methods and properties of the connection object can be used to control synchronization.

### 3.2.1.10 Closing the UltraLite Database Connection

Release software resources when they are no longer being used to prevent the UltraLite database file from remaining in use for as long as the application has a connection to the database.

#### Procedure

1. Call the Close method to release resources.

Use the following code when the application no longer requires a connection to the database:

```
if( conn != NULL ) {  
    conn->Close( &ulerr );  
}
```

2. Call the Fini method to finalize the ULDatabaseManager object.

Use the following code when closing the application.

```
ULDatabaseManager.Fini();
```

#### Results

The database connection is closed and resources are released.

## 3.2.1.11 How to Build and Deploy UltraLite C++ Applications

When building a C++ application that does not use the UltraLite engine, you can either link to a static UltraLite runtime library or, on Windows and Windows Mobile, you can link to an import library and load the UltraLite runtime code dynamically when the application starts.

### Linker/compiler options to build and link runtimes for Linux deployment

The 32-bit linker/compiler options for `libulrt.a` are:

```
-L$SQLANY17/ultralite/linux/x86/lib -lulrt -lulbase
```

The 64-bit linker/compiler options for `libulrt.a` are:

```
-L$SQLANY17/ultralite/linux/x64/lib -lulrt -lulbase
```

The headers command-line option is:

```
-I$SQLANY17/sdk/include
```

#### In this section:

##### [Deploying an UltraLite Application for Microsoft Windows Mobile \(Static Linkage\) \[page 668\]](#)

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, link libraries, method calls, and deployment files to ensure that your UltraLite C++ application runs successfully on Microsoft Windows and Microsoft Windows Mobile devices.

##### [Deploying an UltraLite Application for Microsoft Windows Mobile \(Dynamic Linkage\) \[page 669\]](#)

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, link libraries, method calls, and deployment files to ensure that your UltraLite C++ application runs successfully.

##### [Deploying an UltraLite Application for Microsoft Windows Mobile \(UltraLite Engine\) \[page 671\]](#)

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, link libraries, method calls, and deployment files to ensure that your UltraLite C++ application runs successfully on Microsoft Windows and Microsoft Windows Mobile devices.

##### [Deploying an UltraLite Application for macOS or iOS \[page 673\]](#)

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, method calls, and deployment files to ensure that your UltraLite application runs successfully on Mac computers, iPhones or iPads.

##### [Deploying an UltraLite Application for Linux \[page 675\]](#)

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, method calls, and deployment files to ensure that your UltraLite application runs successfully on Linux.

### 3.2.1.11.1 Deploying an UltraLite Application for Microsoft Windows Mobile (Static Linkage)

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, link libraries, method calls, and deployment files to ensure that your UltraLite C++ application runs successfully on Microsoft Windows and Microsoft Windows Mobile devices.

#### Procedure

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

Synchronization type	Parameter settings
TCP/IP	Set the <code>Stream</code> synchronization parameter to <code>tcpip</code> .
HTTP	Set the <code>Stream</code> synchronization parameter to <code>http</code> .
RSA TLS	Set the <code>Stream</code> synchronization parameter to <code>tls</code> .
RSA HTTPS	Set the <code>Stream</code> synchronization parameter to <code>https</code> .

3. When using RSA end-to-end encryption, set the protocol option `e2ee_public_key= key-file`.
4. When using ZLIB compression, set the protocol option `compression=zlib`.
5. Link against the following files:
  - `ulrt.lib`.
  - `ulbase.lib`.
  - When using RSA TLS, or RSA HTTPS synchronization, `ulrsa.lib`.

For Microsoft Windows Mobile, these files are located in `%SQLANY17%\UltraLite\CE\Arm.50\Lib`. For Microsoft Windows, they are located in `%SQLANY17%\UltraLite\Windows\x64\Lib\VS9` or `%SQLANY17%\UltraLite\Windows\x86\Lib\VS9`.

6. Call the following methods in your UltraLite application:
  - When using AES encryption, the `ULDatabaseManager.EnableAesDBEncryption` method.
7. Ensure that the following methods are called for the synchronization type used in your UltraLite application:

Synchronization type	Parameter settings
TCP/IP	Call the <code>EnableTcpipSynchronization</code> method.
HTTP	Call the <code>EnableHttpSynchronization</code> method.

Synchronization type	Parameter settings
RSA TLS	Call the <code>EnableTlsSynchronization</code> and <code>EnableRsaSyncEncryption</code> methods.
RSA HTTPS	Call the <code>EnableHttpsSynchronization</code> and <code>EnableRsaSyncEncryption</code> methods.

## Results

The UltraLite C++ application, which uses static linkage, runs successfully on the Microsoft Windows desktop or Microsoft Windows Mobile device that it is deployed to.

## Next Steps

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

### 3.2.1.11.2 Deploying an UltraLite Application for Microsoft Windows Mobile (Dynamic Linkage)

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, link libraries, method calls, and deployment files to ensure that your UltraLite C++ application runs successfully.

## Procedure

- Specify the following parameters:
  - When using AES encryption, set the connection parameter `DBKEY= encryption-key` while creating or connecting to the database.
- Set the appropriate parameter settings when using synchronization in your UltraLite application:

Synchronization type	Parameter settings
TCP/IP	Set the <code>Stream</code> synchronization parameter to <code>tcpip</code> .
HTTP	Set the <code>Stream</code> synchronization parameter to <code>http</code> .

Synchronization type	Parameter settings
RSA TLS	Set the <i>Stream</i> synchronization parameter to <code>tls</code> .
RSA HTTPS	Set the <i>Stream</i> synchronization parameter to <code>https</code> .

- When using RSA end-to-end encryption, set the protocol option `e2ee_public_key= key-file`.
- When using ZLIB compression, set the protocol option `compression=zlib`.
- Link against the following files:

- `ulbase.lib`
- `ulimp.lib`

For Microsoft Windows Mobile, these files are located in `%SQLANY17%\UltraLite\CE\Arm.50\Lib`. For Microsoft Windows, they are located in `%SQLANY17%\UltraLite\Windows\x64\Lib\VS9` or `%SQLANY17%\UltraLite\Windows\x86\Lib\VS9`.

- When linking against the `ulimp.lib` library, define the `UL_USE_DLL` preprocessor macro when compiling. For example, specify the following:

```
-DUL_USE_DLL
```

- Call the following methods in your UltraLite application:
  - When using AES encryption, the `ULDatabaseManager.EnableAesDBEncryption` method.
- Ensure that the following methods are called for the synchronization type used in your UltraLite application:

Synchronization type	Parameter settings
TCP/IP	Call the <code>EnableTcpipSynchronization</code> method.
HTTP	Call the <code>EnableHttpSynchronization</code> method.
RSA TLS	Call the <code>EnableTlsSynchronization</code> and <code>EnableRsaSyncEncryption</code> methods.
RSA HTTPS	Call the <code>EnableHttpsSynchronization</code> and <code>EnableRsaSyncEncryption</code> methods.

- Deploy the following files:

- `ulrt17.dll`.
- When using ZLIB compression, `mlczlib17.dll`.
- When using RSA TLS, RSA HTTPS, or RSA E2EE, `mlcrsa17.dll`.

For Microsoft Windows Mobile, the files are located in `%SQLANY17%\UltraLite\CE\Arm.50`. For Microsoft Windows, the files are located in `%SQLANY17%\UltraLite\Windows\x64` or `%SQLANY17%\UltraLite\Windows\x86`.

## Results

The UltraLite C++ application, which uses dynamic linkage, runs successfully on the Microsoft Windows desktop or Microsoft Windows Mobile device that it is deployed to.

## Next Steps

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

### 3.2.1.11.3 Deploying an UltraLite Application for Microsoft Windows Mobile (UltraLite Engine)

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, link libraries, method calls, and deployment files to ensure that your UltraLite C++ application runs successfully on Microsoft Windows and Microsoft Windows Mobile devices.

## Procedure

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

Synchronization type	Parameter settings
TCP/IP	Set the <i>Stream</i> synchronization parameter to tcpip.
HTTP	Set the <i>Stream</i> synchronization parameter to http.
RSA TLS	Set the <i>Stream</i> synchronization parameter to tls.
RSA HTTPS	Set the <i>Stream</i> synchronization parameter to https.

3. When using RSA end-to-end encryption, set the protocol option `e2ee_public_key= key-file`.
4. When using ZLIB compression, set the protocol option `compression=zlib`.
5. Link against the following files:
  - `ulrtc.lib`
  - `ulbase.lib`

For Microsoft Windows Mobile, these files are located in `%SQLANY17%\UltraLite\CE\Arm.50\Lib`. For Microsoft Windows, they are located in `%SQLANY17%\UltraLite\Windows\x64\Lib\VS9` or `%SQLANY17%\UltraLite\Windows\x86\Lib\VS9`.

6. Ensure that the following methods are called for the synchronization type used in your UltraLite application:

Synchronization type	Parameter settings
TCP/IP	Call the <code>EnableTcpipSynchronization</code> method.
HTTP	Call the <code>EnableHttpSynchronization</code> method.
RSA TLS	Call the <code>EnableTlsSynchronization</code> and <code>EnableRsaSyncEncryption</code> methods.
RSA HTTPS	Call the <code>EnableHttpsSynchronization</code> and <code>EnableRsaSyncEncryption</code> methods.

7. Deploy the following files:

- `uleng17.exe`.
- When using ZLIB compression, `mlczlib17.dll`.
- When using RSA TLS, RSA HTTPS, or RSA E2EE `mlcrsa17.dll`.

For Microsoft Windows Mobile, the files are located in `%SQLANY17%\UltraLite\CE\Arm.50`. For Microsoft Windows, the files are located in `%SQLANY17%\UltraLite\Windows\x64` or `%SQLANY17%\UltraLite\Windows\x86`.

## Results

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

## Next Steps

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



### 3.2.1.11.4 Deploying an UltraLite Application for macOS or iOS

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, method calls, and deployment files to ensure that your UltraLite application runs successfully on Mac computers, iPhones or iPads.

#### Procedure

1. For macOS, add the following runtime library files to your Xcode project:
  - `/Applications/SQLAnywhere17/System/ultralite/macosx/x86_64/libulrt.a`
  - `/Applications/SQLAnywhere17/System/ultralite/macosx/x86_64/libulbase.a`
2. For iOS, to link to the UltraLite runtime library, either:

Add `install-dir/ultralite/iphone/libulrt.a` to the *Frameworks* group in Xcode.

**OR**

Add the following to the *Other Linker Flags* (OTHER\_LDFLAGS) build setting:

```
-L$(SQLANY_ROOT)/ultralite/iphone  
-lulrt
```

where `SQLANY_ROOT` is a custom build setting set to the SQL Anywhere installation directory.

UltraLite runtimes must be built after installation. Follow the instructions provided in `install-dir/ultralite/iphone/readme.txt`.

3. Add the appropriate frameworks to your Xcode project:
  - For macOS, `CoreFoundation.framework`, `CoreServices.framework`, and `Security.framework`.
  - For iOS, `CFNetwork.framework` and `Security.framework`.
4. Specify the following parameters:
  - When using AES encryption, set the connection parameter `DBKEY= encryption-key` while creating or connecting to the database.
5. Set the appropriate parameter settings when using synchronization in your UltraLite application:

Synchronization type	Parameter settings
TCP/IP	Set the <i>Stream</i> synchronization parameter to <code>tcpip</code> .
HTTP	Set the <i>Stream</i> synchronization parameter to <code>http</code> .
RSA TLS	Set the <i>Stream</i> synchronization parameter to <code>tls</code> .
RSA HTTPS	Set the <i>Stream</i> synchronization parameter to <code>https</code> .

6. When using RSA E2EE encryption, set the protocol option `e2ee_public_key= key-file`.

7. When using ZLIB compression, set the protocol option `compression=zlib`.
8. When using AES encryption, call the `ULDatabaseManager.EnableAesDBEncryption` method.
9. Ensure that the following methods are called for the synchronization type used in your UltraLite application:

Synchronization type	Parameter settings
TCP/IP	Call the <code>EnableTcpipSynchronization</code> method.
HTTP	Call the <code>EnableHttpSynchronization</code> method.
RSA TLS	Call the <code>EnableTlsSynchronization</code> and <code>EnableRsaSyncEncryption</code> methods.
RSA HTTPS	Call the <code>EnableHttpsSynchronization</code> and <code>EnableRsaSyncEncryption</code> methods.

## Results

The UltraLite application runs successfully on the macOS desktop or iOS device that it is deployed to.

## Next Steps

Deploy an UltraLite database to the Mac desktop, iPhone, or iPad that the application was deployed to, or create a new database with the deployed application.

## Related Information

[Apple iOS and macOS Considerations \[page 646\]](#)

## 3.2.1.11.5 Deploying an UltraLite Application for Linux

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, method calls, and deployment files to ensure that your UltraLite application runs successfully on Linux.

### Procedure

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

Synchronization type	Parameter settings
TCP/IP	Set the <i>Stream</i> synchronization parameter to <code>tcpip</code> .
HTTP	Set the <i>Stream</i> synchronization parameter to <code>http</code> .
RSA TLS	Set the <i>Stream</i> synchronization parameter to <code>tls</code> .
RSA HTTPS	Set the <i>Stream</i> synchronization parameter to <code>https</code> .

3. When using RSA or FIPS 140-2 RSA end-to-end encryption, set the protocol option `e2ee_public_key= key-file`.
4. When using ZLIB compression, set the protocol option `compression=zlib`.
5. Link against the following files:
  - `libulrt.a`.
  - `libulbase.a`.
  - When using RSA TLS, RSA HTTPS, or RSA E2EE, `libulrsa.a`.

These files are located in `/opt/sqlanywhere17/ultralite/linux/x86/586/lib`.

6. When using AES encryption, call the `ULDatabaseManager.EnableAesDBEncryption` method.
7. Ensure that the following methods are called for the synchronization type used in your UltraLite application:

Synchronization type	Parameter settings
TCP/IP	Call the <code>EnableTcpipSynchronization</code> method.
HTTP	Call the <code>EnableHttpSynchronization</code> method.
RSA TLS	Call the <code>EnableTlsSynchronization</code> and <code>EnableRsaSyncEncryption</code> methods.

Synchronization type	Parameter settings
RSA HTTPS	Call the EnableHttpsSynchronization and EnableRsaSyncEncryption methods.

## Results

The UltraLite application runs successfully on the Linux computer that it is deployed to.

## Next Steps

Deploy an UltraLite database to the Linux computer that the application was deployed to, or create a new database with the deployed application.

## 3.2.2 UltraLite C++ Application Development Using Embedded SQL

You can write database access codes for Embedded SQL UltraLite applications.

### In this section:

#### [Quick Start Guide to UltraLite Embedded SQL Application Development \[page 677\]](#)

When developing Embedded SQL applications, you mix SQL statements with standard C or C++ source code. To develop Embedded SQL applications you should be familiar with the C or C++ programming language.

#### [Example of Embedded SQL \[page 678\]](#)

Embedded SQL is an environment that is a combination of C/C++ program code and pseudo-code.

#### [SQL Communications Area Initialization \[page 680\]](#)

The SQL Communications Area (SQLCA) is an area of memory that is used for communicating statistics and errors from the application to the database and back to the application. The SQLCA is used as a handle for the application-to-database communication link.

#### [UltraLite Database Connection Using Embedded SQL \[page 682\]](#)

To connect to an UltraLite database from an Embedded SQL application, include the EXEC SQL CONNECT statement in your code after initializing the SQLCA.

#### [Host Variables \[page 683\]](#)

Embedded SQL applications use host variables to communicate values to and from the database. Host variables are C variables that are identified to the SQL preprocessor in a declaration section.

#### [Data Fetching \[page 694\]](#)

Fetching data in Embedded SQL is done using the SELECT statement.

#### [User Authentication \[page 697\]](#)

User authentication can be controlled using the `ULGrantConnectTo` and `ULRevokeConnectFrom` methods.

[Data Encryption with UltraLite Embedded SQL \[page 698\]](#)

You can encrypt or obfuscate your UltraLite database using UltraLite Embedded SQL.

[Synchronization Setup for an Embedded SQL Application \[page 699\]](#)

Synchronization is a key feature of many UltraLite applications. Members of the structures in the Embedded SQL API are similar to the UltraLite C++ API.

[Embedded SQL Application Building \[page 706\]](#)

Building an Embedded SQL application requires knowledge of several configuration settings.

## Related Information

[UltraLite Embedded SQL API Reference \[page 731\]](#)

### 3.2.2.1 Quick Start Guide to UltraLite Embedded SQL Application Development

When developing Embedded SQL applications, you mix SQL statements with standard C or C++ source code. To develop Embedded SQL applications you should be familiar with the C or C++ programming language.

The development process for Embedded SQL applications is as follows:

1. Design your UltraLite database.
2. Write your source code in an Embedded SQL source file, which typically has extension `.sqlc`.  
When you need data access in your source code, use the SQL statement you want to execute, prefixed by the `EXEC SQL` keywords. For example:

```
EXEC SQL BEGIN DECLARE SECTION
    int cost
    char pname[31];
EXEC SQL END DECLARE SECTION
EXEC SQL SELECT price, prod_name
    INTO :cost, :pname
    FROM ULProduct
    WHERE prod_id= :pid;
```

3. Preprocess the `.sqlc` files.  
SQL Anywhere includes a SQL preprocessor (`sqlpp`), which reads the `.sqlc` files and generates `.cpp` files. These files hold function calls to the UltraLite runtime library.
4. Compile your `.cpp` files.
5. Link the `.cpp` files.  
You must link the files with the UltraLite runtime library.

## Related Information

[Embedded SQL Application Building \[page 706\]](#)

### 3.2.2.2 Example of Embedded SQL

Embedded SQL is an environment that is a combination of C/C++ program code and pseudo-code.

The pseudo-code that can be interspersed with traditional C/C++ code is a subset of SQL statements. A preprocessor converts the Embedded SQL statements into function calls that are part of the actual code that is compiled to create the application.

Following is a very simple example of an Embedded SQL program. It illustrates updating an UltraLite database record by changing the surname of employee 195.

```
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
main( )
{
    db_init( &sqlca );
    EXEC SQL WHENEVER SQLERROR GOTO error;
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
    EXEC SQL UPDATE employee
        SET emp_lname = 'Johnson'
        WHERE emp_id = 195;
    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
    return( 0 );
error:
    printf( "update unsuccessful: sqlcode = %ld\n",
        sqlca.sqlcode );
    return( -1 );
}
```

Although this example is too simplistic to be useful, it illustrates the following aspects common to all Embedded SQL applications:

- Each SQL statement is prefixed with the keywords EXEC SQL.
- Each SQL statement ends with a semicolon.
- Some Embedded SQL statements are not part of standard SQL. The INCLUDE SQLCA statement is one example.
- In addition to SQL statements, Embedded SQL also provides library functions to perform some specific tasks. The functions db\_init and db\_fini are two examples of library function calls.

## Initialization

The above sample code illustrates initialization statements that must be included before working with the data in an UltraLite database:

1. Define the SQL Communications Area (SQLCA), using the following command:

```
EXEC SQL INCLUDE SQLCA;
```

This definition must be the first Embedded SQL statement, so a natural place for it is the end of the include list.

If you have multiple .sql files in your application, each file must have this line.

2. The first database action must be a call to an Embedded SQL library function named `db_init`. This function initializes the UltraLite runtime library. Only Embedded SQL definition statements can be executed before this call.
3. You must use the SQL CONNECT statement to connect to the UltraLite database.

## Preparing to Exit

The above sample code demonstrates the sequence of calls required when preparing to exit:

1. Commit or rollback any outstanding changes.
2. Disconnect from the database.
3. End your SQL work with a call to a library method named `db_fini`.

When you exit, any uncommitted database changes are automatically rolled back.

## Error Handling

There is virtually no interaction between the SQL and C code in this example. The C code only controls the flow of the program. The WHENEVER statement is used for error checking. The error action, GOTO in this example, is executed whenever any SQL statement causes an error.

**In this section:**

[Embedded SQL Program Structure \[page 680\]](#)

All Embedded SQL statements start with the words EXEC SQL and end with a semicolon.

## Related Information

[db\\_init Method \[page 733\]](#)

### 3.2.2.2.1 Embedded SQL Program Structure

All Embedded SQL statements start with the words EXEC SQL and end with a semicolon.

Normal C-language comments are allowed in the middle of Embedded SQL statements.

Every C program using Embedded SQL must contain the following statement before any other Embedded SQL statements in the source file.

```
EXEC SQL INCLUDE SQLCA;
```

The first Embedded SQL executable statement in the program must be a SQL CONNECT statement. The CONNECT statement supplies connection parameters that are used to establish a connection to the UltraLite database.

Some Embedded SQL commands do not generate any executable C code, or do not involve communication with the database. Only these commands are allowed before the CONNECT statement. Most notable are the INCLUDE statement and the WHENEVER statement for specifying error processing.

### 3.2.2.3 SQL Communications Area Initialization

The SQL Communications Area (SQLCA) is an area of memory that is used for communicating statistics and errors from the application to the database and back to the application. The SQLCA is used as a handle for the application-to-database communication link.

The SQLCA is passed explicitly to all database library functions that communicate with the database. It is implicitly passed in all Embedded SQL statements.

UltraLite defines a SQLCA global variable for you in the generated code. The preprocessor generates an external reference for the global SQLCA variable. The external reference is named sqlca and is of type SQLCA. The actual global variable is declared in the import library.

The SQLCA type is defined in the header file `%SQLANYI7%\SDK\Include\sqlca.h`.

After declaring the SQLCA (`EXEC SQL INCLUDE SQLCA;`), but before your application can perform any operations on a database, you must initialize the communications area by calling `db_init` and passing it the SQLCA:

```
db_init( &sqlca );
```

### SQLCA Provides Error Codes

You reference the SQLCA to test for a particular error code. The `sqlcode` field contains an error code when a database request causes an error. Macros are defined for referencing the `sqlcode` field and some other fields in the `sqlca`.

**In this section:**

[SQLCA Fields Used in UltraLite C++ \[page 681\]](#)



The SQLCA contains the several fields that describe the SQLCA structure.

### 3.2.2.3.1 SQLCA Fields Used in UltraLite C++

The SQLCA contains the several fields that describe the SQLCA structure.

#### **sqlcaid**

An 8-byte character field that contains the string SQLCA as an identification of the SQLCA structure. This field helps in debugging when you are looking at memory contents.

#### **sqlcab**

A long integer that contains the length in bytes of the SQLCA structure.

#### **sqlcode**

A long integer that contains an error code when the database detects an error on a request. Definitions for the error codes are in the header file `%SQLANY17%\SDK\Include\sqlerr.h`. The error code is 0 (zero) for a successful operation, a positive value for a warning, and a negative value for an error.

You can access this field directly using the SQLCODE macro.

#### **sqlerrml**

The length of the information in the sqlerrmc field.

UltraLite applications do not use this field.

#### **sqlerrmc**

May contain one or more character strings to be inserted into an error message. Some error messages contain a placeholder string (`%1`) which is replaced with the text in this field.

UltraLite applications do not use this field.

#### **sqlerrp**

Reserved.

#### **sqlerrd**

A utility array of long integers.

#### **sqlwarn**

Reserved.

UltraLite applications do not use this field.

#### **sqlstate**

The SQLSTATE status value.

UltraLite applications do not use this field.

## 3.2.2.4 UltraLite Database Connection Using Embedded SQL

To connect to an UltraLite database from an Embedded SQL application, include the EXEC SQL CONNECT statement in your code after initializing the SQLCA.

The CONNECT statement has the following form:

```
EXEC SQL CONNECT USING  
'uid=user-name;pwd=password;dbf=database-filename';
```

The connection string (enclosed in single quotes) may include additional database connection parameters.

If you want more than one database connection in your application, you can either use multiple SQLCAs or you can use a single SQLCA to manage the connections.

### Use a Single SQLCA

You can use a single SQLCA to manage multiple connections to a database.

Each SQLCA has a single active or current connection, but that connection can be changed. Before executing a command, use the SET CONNECTION statement to specify the connection on which the command should be executed.

**In this section:**

[Using Multiple SQLCAs to Manage Multiple Database Connections \[page 682\]](#)

Use the SET SQLCA Embedded SQL statement to tell the SQL preprocessor to use a specific SQLCA for database requests.

### 3.2.2.4.1 Using Multiple SQLCAs to Manage Multiple Database Connections

Use the SET SQLCA Embedded SQL statement to tell the SQL preprocessor to use a specific SQLCA for database requests.

#### Procedure

1. Initialize Each SQLCA used in your program with a call to db\_init.
2. At the top of your program or in a header file, set the SQLCA reference to point at task specific data:

```
EXEC SQL SET SQLCA 'task_data->sqlca';
```

This statement does not generate any code and does not affect performance.

## Results

The state within the preprocessor changes so that any reference to the SQLCA uses the given string.

## Next Steps

Clean up resources when closing your application with a call to the `db_fini` method.

## Related Information

[db\\_init Method \[page 733\]](#)

### 3.2.2.5 Host Variables

Embedded SQL applications use host variables to communicate values to and from the database. Host variables are C variables that are identified to the SQL preprocessor in a declaration section.

#### In this section:

[Host Variable Declaration \[page 684\]](#)

Define host variables by placing them within a declaration section. Host variables are declared by surrounding the normal C variable declarations with `BEGIN DECLARE SECTION` and `END DECLARE SECTION` statements.

[Data Types \[page 684\]](#)

To transfer information between a program and the database server, every data item must have a data type. You can create a host variable with any one of the supported types.

[Host Variable Usage in UltraLite C++ \[page 687\]](#)

Host variables can be used in the several specific circumstances.

[Host Variable Scope \[page 688\]](#)

A host-variable declaration section can appear anywhere that C variables can normally be declared, including the parameter declaration section of a C function.

[Expressions as Host Variables \[page 689\]](#)

Host variables must be simple names because the SQL preprocessor does not recognize pointer or reference expressions.

[Host Variables in C++ \[page 690\]](#)

It is convenient to frequently declare your class in a separate header file.

[Indicator Variables \[page 692\]](#)

Indicator variables are C variables that hold supplementary information about a particular host variable. You can use a host variable when fetching or putting data. Use indicator variables to handle NULL values.

### 3.2.2.5.1 Host Variable Declaration

Define host variables by placing them within a declaration section. Host variables are declared by surrounding the normal C variable declarations with `BEGIN DECLARE SECTION` and `END DECLARE SECTION` statements.

Whenever you use a host variable in a SQL statement, you must prefix the variable name with a colon (`:`) so the SQL preprocessor knows you are referring to a (declared) host variable and distinguish it from other identifiers allowed in the statement.

You can use host variables in place of value constants in any SQL statement. When the database server executes the command, the value of the host variable is read from or written to each host variable. Host variables cannot be used in place of table or column names.

The SQL preprocessor does not scan C language code except inside a declaration section. Initializers for variables are allowed inside a declaration section, while typedef types and structures are not permitted.

The following sample code illustrates the use of host variables with an `INSERT` command. The variables are filled in by the program and then inserted into the database:

```
/* Declare fields for personal data. */
EXEC SQL BEGIN DECLARE SECTION;
    long employee_number = 0;
    char employee_name[50];
    char employee_initials[8];
    char employee_phone[15];
EXEC SQL END DECLARE SECTION;
/* Fill variables with appropriate values. */
/* Insert a row in the database. */
EXEC SQL INSERT INTO Employee
    VALUES (:employee_number, :employee_name,
            :employee_initials, :employee_phone );
```

### 3.2.2.5.2 Data Types

To transfer information between a program and the database server, every data item must have a data type. You can create a host variable with any one of the supported types.

Only a limited number of C data types are supported as host variables. Also, certain host variable types do not have a corresponding C type.

Macros defined in the `sqlca.h` header file can be used to declare a host variable of type `VARCHAR`, `FIXCHAR`, `BINARY`, `DECIMAL`, or `SQLDATETIME`. These macros are used as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
    DECL_VARCHAR( 10 ) v_varchar;
    DECL_FIXCHAR( 10 ) v_fixchar;
    DECL_BINARY( 4000 ) v_binary;
    DECL_DECIMAL( 10, 2 ) v_packed_decimal;
    DECL_DATETIME v_datetime;
EXEC SQL END DECLARE SECTION;
```

The preprocessor recognizes these macros within a declaration section and treats the variable as the appropriate type.

The following data types are supported by the Embedded SQL programming interface:

#### 16-bit signed integer

```
short int I;
unsigned short int I;
```

#### 32-bit signed integer

```
long int l;
unsigned long int l;
```

#### 4-byte floating-point number

```
float f;
```

#### 8-byte floating-point number

```
double d;
```

#### Packed decimal number

```
DECL_DECIMAL(p,s)
typedef struct TYPE_DECIMAL {
    char array[1];
} TYPE_DECIMAL;
```

#### Null terminated, blank-padded character string

```
char a[n]; /* n > 1 */
char *a; /* n = 2049 */
```

Because the C-language array must also hold the NULL terminator, a char a[n] data type maps to a CHAR(n - 1) SQL data type, which can hold -1 characters.

#### i Note

The SQL preprocessor assumes that a **pointer to char** points to a character array of size 2049 bytes and that this array can safely hold 2048 characters, plus the NULL terminator. In other words, a char\* data type maps to a CHAR(2048) SQL type. If that is not the case, your application may corrupt memory.

If you are using a 16-bit compiler, requiring 2049 bytes can make the program stack overflow. Instead, use a declared array, even as a parameter to a function, to let the SQL preprocessor know the size of the array. WCHAR and TCHAR behave similarly to char.

#### NULL terminated UNICODE or wide character string

Each character occupies two bytes of space and so may contain UNICODE characters.

```
WCHAR a[n]; /* n > 1 */
```

#### NULL terminated system-dependent character string

A TCHAR is equivalent to a WCHAR for systems that use UNICODE (for example, Microsoft Windows Mobile) for their character set; otherwise, a TCHAR is equivalent to a char. The TCHAR data type is designed to support character strings in either kind of system automatically.

```
TCHAR a[n]; /* n > 1 */
```

### Fixed-length blank padded character string

```
char a; /* n = 1 */
DECL_FIXCHAR(n) a; /* n >= 1 */
```

### Variable-length character string with a two-byte length field

When supplying information to the database server, you must set the length field. When fetching information from the database server, the server sets the length field (not padded).

```
DECL_VARCHAR(n) a; /* n >= 1 */
typedef struct VARCHAR {
    a_sql_ulen len;
    TCHAR array[1];
} VARCHAR;
```

### Variable-length binary data with a two-byte length field

When supplying information to the database server, you must set the length field. When fetching information from the database server, the server sets the length field.

```
DECL_BINARY(n) a; /* n >= 1 */
typedef struct BINARY {
    a_sql_ulen len;
    unsigned char array[1];
} BINARY;
```

### SQLDATETIME structure with fields for each part of a timestamp

```
DECL_DATETIME a;
typedef struct SQLDATETIME {
    unsigned short year; /* for example: 1999 */
    unsigned char month; /* 0-11 */
    unsigned char day_of_week; /* 0-6, 0 = Sunday */
    unsigned short day_of_year; /* 0-365 */
    unsigned char day; /* 1-31 */
    unsigned char hour; /* 0-23 */
    unsigned char minute; /* 0-59 */
    unsigned char second; /* 0-59 */
    unsigned long microsecond; /* 0-999999 */
} SQLDATETIME;
```

The SQLDATETIME structure is used to retrieve fields of the DATE, TIME, and TIMESTAMP type (or anything that can be converted to one of these). Often, applications have their own formats and date manipulation code. Fetching data in this structure makes it easier for you to manipulate this data. DATE, TIME, and TIMESTAMP fields can also be fetched and updated with any character type.

If you use a SQLDATETIME structure to enter a date, time, or timestamp into the database, the day\_of\_year and day\_of\_week members are ignored.

### DT\_LONGVARCHAR

Long varying length character data. The macro defines a structure, as follows:

```
#define DECL_LONGVARCHAR( size ) \
    struct { a_sql_uint32    array_len;    \
            a_sql_uint32    stored_len;   \
            a_sql_uint32    untrunc_len;  \
            char             array[size+1];\
    }
```

The DECL\_LONGVARCHAR struct can be used with more than 32KB of data. Data can be fetched all at once, or in pieces using the GET DATA statement. Data can be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement. The data is not null terminated.

### DT\_LONGBINARY

Long binary data. The macro defines a structure, as follows:

```
#define DECL_LONGBINARY( size ) \
    struct { a_sql_uint32      array_len;      \
             a_sql_uint32      stored_len;     \
             a_sql_uint32      untrunc_len;    \
             char               array[size];    \
    }
```

The DECL\_LONGBINARY struct can be used with more than 32KB of data. Data can be fetched all at once, or in pieces using the GET DATA statement. Data can be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement.

The structures are defined in the `%SQLANY17%\SDK\Include\sqlca.h` file. The VARCHAR, BINARY, and TYPE\_DECIMAL types contain a one-character array and are not useful for declaring host variables. However, they are useful for allocating variables dynamically or typecasting other variables.

## DATE and TIME Database Types

There are no corresponding Embedded SQL interface data types for the various DATE and TIME database types. These database types are fetched and updated either using the SQLDATETIME structure or using character strings.

There are no Embedded SQL interface data types for LONG VARCHAR and LONG BINARY database types.

### 3.2.2.5.3 Host Variable Usage in UltraLite C++

Host variables can be used in the several specific circumstances.

- In a SELECT, INSERT, UPDATE, or DELETE statement in any place where a number or string constant is allowed.
- In the INTO clause of a SELECT or FETCH statement.
- In CONNECT, DISCONNECT, and SET CONNECT statements, a host variable can be used in place of a user ID, password, connection name, or database name.

Host variables can *never* be used in place of a table name or a column name.

### 3.2.2.5.4 Host Variable Scope

A host-variable declaration section can appear anywhere that C variables can normally be declared, including the parameter declaration section of a C function.

The C variables have their normal scope (available within the block in which they are defined). However, since the SQL preprocessor does not scan C code, it does not respect C blocks.

#### Preprocessor Assumes All Host Variables Are Global

As far as the SQL preprocessor is concerned, host variables are globally known in the source module following their declaration. Two host variables cannot have the same name. The only exception to this rule is that two host variables can have the same name if they have identical types (including any necessary lengths).

The best practice is to give each host variable a unique name.

#### Example

Because the SQL preprocessor cannot parse C code, it assumes all host variables, no matter where they are declared, are known globally following their declaration.

```
// Example demonstrating poor coding
EXEC SQL BEGIN DECLARE SECTION;
    long emp_id;
EXEC SQL END DECLARE SECTION;
long getManagerID( void )
{
    EXEC SQL BEGIN DECLARE SECTION;
        long manager_id = 0;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SELECT manager_id
        INTO :manager_id
        FROM employee
        WHERE emp_number = :emp_id;
    return( manager_number );
}
void setManagerID( long manager_id )
{
    EXEC SQL UPDATE employee
        SET manager_number = :manager_id
        WHERE emp_number = :emp_id;
}
```

Although the above code works, it is confusing because the SQL preprocessor relies on the declaration inside `getManagerID` when processing the statement within `setManagerID`. Rewrite this code as follows:

```
// Rewritten example
#if 0
    // Declarations for the SQL preprocessor
    EXEC SQL BEGIN DECLARE SECTION;
        long emp_id;
        long manager_id;
    EXEC SQL END DECLARE SECTION;
```



```

#endif
long getManagerID( long emp_id )
{
    long manager_id = 0;
    EXEC SQL SELECT manager_id
              INTO :manager_id
              FROM employee
              WHERE emp_number = :emp_id;
    return( manager_number );
}
void setManagerID( long emp_id, long manager_id )
{
    EXEC SQL UPDATE employee
              SET manager_number = :manager_id
              WHERE emp_number = :emp_id;
}

```

The SQL preprocessor sees the declaration of the host variables contained within the `#if` directive because it ignores these directives. However, it ignores the declarations within the procedures because they are not inside a `DECLARE SECTION`. Conversely, the C compiler ignores the declarations within the `#if` directive and uses those within the procedures.

These declarations work only because variables having the same name are declared to have exactly the same type.

### 3.2.2.5.5 Expressions as Host Variables

Host variables must be simple names because the SQL preprocessor does not recognize pointer or reference expressions.

For example, the following statement *does not work* because the SQL preprocessor does not understand the dot operator. The same syntax has a different meaning in SQL.

```

// Incorrect statement:
EXEC SQL SELECT LAST sales_id INTO :mystruct.mymember;

```

Although the above syntax is not allowed, you can still use an expression with the following technique:

- Wrap the SQL declaration section in an `#if 0` preprocessor directive. The SQL preprocessor will read the declarations and use them for the rest of the module because it ignores preprocessor directives.
- Define a macro with the same name as the host variable. Since the SQL declaration section is not seen by the C compiler because of the `#if` directive, no conflict will arise. Ensure that the macro evaluates to the same type host variable.

The following code demonstrates this technique to hide the `host_value` expression from the SQL preprocessor.

```

#include <sqlerr.h>
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
typedef struct my_struct {
    long    host_field;
} my_struct;
#if 0
    // Because it ignores #if preprocessing directives,
    // SQLPP reads the following declaration.
EXEC SQL BEGIN DECLARE SECTION;

```

```

        long    host_value;
    EXEC SQL END DECLARE SECTION;
#endif
// Make C/C++ recognize the 'host_value' identifier
// as a macro that expands to a struct field.
#define host_value my_s.host_field

```

Since the SQLPP processor ignores directives for conditional compilation, `host_value` is treated as a long host variable and will emit that name when it is subsequently used as a host variable. The C/C++ compiler processes the emitted file and will substitute `my_s.host_field` for all such uses of that name.

With the above declarations in place, you can proceed to access `host_field` as follows.

```

void main( void )
{
    my_struct    my_s;
    db_init( &sqlca );
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "SQL";
    EXEC SQL DECLARE my_table_cursor CURSOR FOR
        SELECT int_col FROM my_table order by int_col;
    EXEC SQL OPEN my_table_cursor;
    for( ; ; ) {
        // :host_value references my_s.host_field
        EXEC SQL FETCH NEXT AllRows INTO :host_value;
        if( SQLCODE == SQLE_NOTFOUND ) {
            break;
        }
        printf( "%ld\n", my_s.host_field );
    }
    EXEC SQL CLOSE my_table_cursor;
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
}

```

You can use the same technique to use other lvalues as host variables:

- pointer indirections

```

*ptr
p_struct->ptr
(*pp_struct)->ptr

```

- array references

```

my_array[ I ]

```

- arbitrarily complex lvalues

### 3.2.2.5.6 Host Variables in C++

It is convenient to frequently declare your class in a separate header file.

This header file might contain, for example, the following declaration of `my_class`.

```

typedef short a_bool;
#define TRUE ((a_bool)(1==1))
#define FALSE ((a_bool)(0==1))
public class {
    long host_member;
    my_class(); // Constructor

```

```

~my_class();          // Destructor
a_bool FetchNextRow( void );
    // Fetch the next row into host_member
} my_class;

```

In this example, each method is implemented in an Embedded SQL source file. Only simple variables can be used as host variables. The technique introduced in the preceding section can be used to access a data member of a class.

```

EXEC SQL INCLUDE SQLCA;
#include "my_class.hpp"
#if 0
    // Because it ignores #if preprocessing directives,
    // SQLPP reads the following declaration.
    EXEC SQL BEGIN DECLARE SECTION;
        long this_host_member;
    EXEC SQL END DECLARE SECTION;
#endif
// Macro used by the C++ compiler only.
#define this_host_member this->host_member
my_class::my_class()
{
    EXEC SQL DECLARE my_table_cursor CURSOR FOR
        SELECT int_col FROM my_table order by int_col;
    EXEC SQL OPEN my_table_cursor;
}
my_class::~my_class()
{
    EXEC SQL CLOSE my_table_cursor;
}
a_bool my_class::FetchNextRow( void )
{
    // :this_host_member references this->host_member
    EXEC SQL FETCH NEXT AllRows INTO :this_host_member;
    return( SQLCODE != SQLE_NOTFOUND );
}
void main( void )
{
    db_init( &sqlca );
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "SQL";
    {
        my_class mc; // Created after connecting.
        while( mc.FetchNextRow() ) {
            printf( "%ld\n", mc.host_member );
        }
    }
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
}

```

The above example declares `this_host_member` for the SQL preprocessor, but the macro causes C++ to convert it to `this->host_member`. The preprocessor would otherwise not know the type of this variable. Many C/C++ compilers do not tolerate duplicate declarations. The `#if` directive hides the second declaration from the compiler, but leaves it visible to the SQL preprocessor.

While multiple declarations can be useful, you must ensure that each declaration assigns the same variable name to the same type. The preprocessor assumes that each host variable is globally known following its declaration because it cannot fully parse the C language.

## 3.2.2.5.7 Indicator Variables

Indicator variables are C variables that hold supplementary information about a particular host variable. You can use a host variable when fetching or putting data. Use indicator variables to handle NULL values.

An indicator variable is a host variable of type `a_sql_len` that is placed immediately following a regular host variable in a SQL statement. To detect or specify a NULL value, place the indicator variable immediately following a regular host variable in a SQL statement.

### Example

For example, in the following INSERT statement, `:ind_phone` is an indicator variable.

```
EXEC SQL INSERT INTO Employee
VALUES (:employee_number, :employee_name,
       :employee_initials, :employee_phone:ind_phone );
```

On a fetch or execute where no rows are received from the database server (such as when an error or end of result set occurs), then indicator values are unchanged.

#### i Note

To allow for the future use of 32 and 64-bit lengths and indicators, the use of short int for Embedded SQL indicator variables is deprecated. Use `a_sql_len` instead.

### Indicator Variable Values

The following table provides a summary of indicator variable usage:

Indicator value	Supplying value to database	Receiving value from database
0	Host variable value	Fetches a non-NULL value.
-1	NULL value	Fetches a NULL value

#### In this section:

##### [Indicator Variables to Handle NULL \[page 693\]](#)

Do not confuse the SQL concept of NULL with the C-language constant of the same name. In the SQL language, NULL represents either an unknown attribute or inapplicable information. The C-language constant represents a pointer value that does not point to a memory location.

### 3.2.2.5.7.1 Indicator Variables to Handle NULL

Do not confuse the SQL concept of NULL with the C-language constant of the same name. In the SQL language, NULL represents either an unknown attribute or inapplicable information. The C-language constant represents a pointer value that does not point to a memory location.

When NULL is used in the SQL Anywhere documentation, it refers to the SQL database meaning given above. The C language constant is referred to as the null pointer (lowercase).

NULL is not the same as any value of the column's defined type. Indicator variables are needed to pass NULL values to the database or receive NULL results back.

### Using Indicator Variables When Inserting NULL

An INSERT statement can include an indicator variable as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
short int employee_number;
char employee_name[50];
char employee_initials[6];
char employee_phone[15];
a_sql len ind_phone;
EXEC SQL END DECLARE SECTION;
/* set values of employee number, name,
   initials, and phone number */
if( /* phone number is known */ ) {
    ind_phone = 0;
} else {
    ind_phone = -1; /* NULL */
}
EXEC SQL INSERT INTO Employee
VALUES (:employee_number, :employee_name,
       :employee_initials, :employee_phone:ind_phone );
```

If the indicator variable has a value of -1, a NULL is written. If it has a value of 0, the actual value of employee\_phone is written.

### Using Indicator Variables When Fetching NULL

Indicator variables are also used when receiving data from the database. They are used to indicate that a NULL value was fetched (indicator is negative). If a NULL value is fetched from the database and an indicator variable is not supplied, the SQLE\_NO\_INDICATOR error is generated.

### Related Information

[SQL Communications Area Initialization \[page 680\]](#)

## 3.2.2.6 Data Fetching

Fetching data in Embedded SQL is done using the SELECT statement.

There are two cases that can be returned:

1. The SELECT statement returns no rows or returns exactly one row.
2. The SELECT statement returns multiple rows.

**In this section:**

[Single Row Fetching \[page 694\]](#)

A single row query retrieves at most one row from the database.

[Multiple Row Fetching \[page 695\]](#)

You use a cursor to retrieve rows from a query that has multiple rows in the result set. A cursor is a handle or an identifier for the SQL query result set and a position within that result set.

### 3.2.2.6.1 Single Row Fetching

A single row query retrieves at most one row from the database.

A single row query SELECT statement may have an INTO clause following the select list and before the FROM clause. The INTO clause contains a list of host variables to receive the value for each select list item. There must be the same number of host variables as there are select list items. The host variables may be accompanied by indicator variables to indicate NULL results.

When the SELECT statement is executed, the database server retrieves the results and places them in the host variables.

- If the query returns more than one row, the database server returns the SQLE\_TOO\_MANY\_RECORDS error.
- If the query returns no rows, the SQLE\_NOTFOUND warning is returned.

### Example

For example, the following code fragment returns 1 if a row from the employee table is successfully fetched, 0 if the row doesn't exist, and -1 if an error occurs.

```
EXEC SQL BEGIN DECLARE SECTION;
    long int    emp_id;
    char        name[41];
    char        sex;
    char        birthdate[15];
    a_sql_len   ind birthdate;
EXEC SQL END DECLARE SECTION;
int find_employee( long employee )
{
    emp_id = employee;
    EXEC SQL SELECT emp_fname || ' ' || emp_lname,
        sex, birth_date
```

```

        INTO :name, :sex, birthdate:ind_birthdate
        FROM "DBA".employee
        WHERE emp_id = :emp_id;
    if( SQLCODE == SQLE_NOTFOUND ) {
        return( 0 ); /* employee not found */
    } else if( SQLCODE < 0 ) {
        return( -1 ); /* error */
    } else {
        return( 1 ); /* found */
    }
}

```

## Related Information

[SQL Communications Area Initialization \[page 680\]](#)

### 3.2.2.6.2 Multiple Row Fetching

You use a cursor to retrieve rows from a query that has multiple rows in the result set. A cursor is a handle or an identifier for the SQL query result set and a position within that result set.

Cursors in UltraLite applications are always opened using the WITH HOLD option. They are never closed automatically. You must explicitly close each cursor using the CLOSE statement.

A cursor can be managed using the following steps:

1. Declare a cursor for a particular SELECT statement, using the DECLARE statement.
2. Open the cursor using the OPEN statement.
3. Retrieve rows from the cursor one at a time using the FETCH statement.
4. Fetch rows until the SQLE\_NOTFOUND warning is returned. Error and warning codes are returned in the variable SQLCODE, defined in the SQL communications area structure.
5. Close the cursor, using the CLOSE statement.

The following is a simple example of cursor usage:

```

void print_employees( void )
{
    int status;
    EXEC SQL BEGIN DECLARE SECTION;
    char name[50];
    char sex;
    char birthdate[15];
    a_sql_len ind_birthdate;
    EXEC SQL END DECLARE SECTION;
    /* 1. Declare the cursor. */
    EXEC SQL DECLARE C1 CURSOR FOR
        SELECT emp_fname || ' ' || emp_lname,
               sex, birth_date
        FROM "DBA".employee
        ORDER BY emp_fname, emp_lname;
    /* 2. Open the cursor. */
    EXEC SQL OPEN C1;
    /* 3. Fetch each row from the cursor. */
    for( ;; ) {
        EXEC SQL FETCH C1 INTO :name, :sex,

```

```

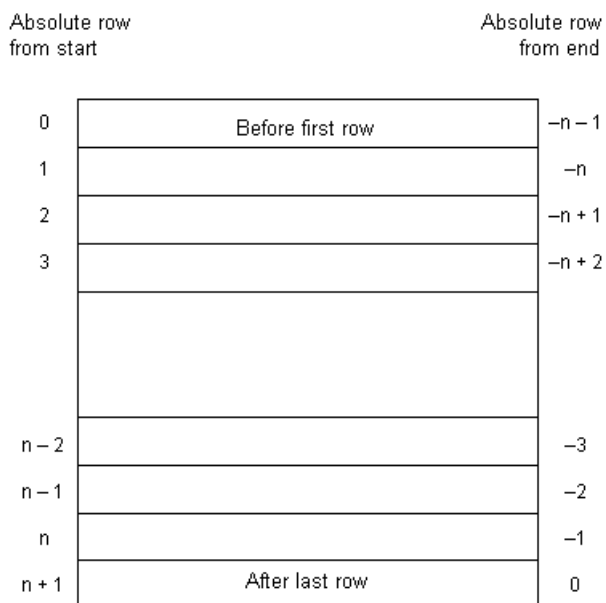
        :birthdate:ind_birthdate;
    if( SQLCODE == SQLE_NOTFOUND ) {
        break; /* no more rows */
    } else if( SQLCODE < 0 ) {
        break; /* the FETCH caused an error */
    }
    if( ind_birthdate < 0 ) {
        strcpy( birthdate, "UNKNOWN" );
    }
    printf( "Name: %s Sex: %c Birthdate:
            %s\n",name, sex, birthdate );
}
/* 4. Close the cursor. */
EXEC SQL CLOSE C1;
}

```

## Cursor Positioning

A cursor is positioned in one of three places:

- On a row
- Before the first row
- After the last row



## Order of Rows in a Cursor

You control the order of rows in a cursor by including an ORDER BY clause in the SELECT statements that defines that cursor. If you omit this clause, the order of the rows is unpredictable.



If you don't explicitly define an order, the only guarantee is that fetching repeatedly will return each row in the result set once and only once before `SQLE_NOTFOUND` is returned.

## Repositioning a Cursor

When you open a cursor, it is positioned before the first row. The `FETCH` statement automatically advances the cursor position. An attempt to `FETCH` beyond the last row results in a `SQLE_NOTFOUND` error, which can be used as a convenient signal to complete sequential processing of the rows.

You can also reposition the cursor to an absolute position relative to the start or end of the query results, or you can move the cursor relative to the current position. There are special *positioned* versions of the `UPDATE` and `DELETE` statements that can be used to update or delete the row at the current position of the cursor. If the cursor is positioned before the first row or after the last row, a `SQLE_NOTFOUND` error is returned.

To avoid unpredictable results when using explicit positioning, you can include an `ORDER BY` clause in the `SELECT` statement that defines the cursor.

You can use the `PUT` statement to insert a row into a cursor.

## Cursor Positioning After Updates

After updating any information that is being accessed by an open cursor, it is best to fetch and display the rows again. If the cursor is being used to display a single row, `FETCH RELATIVE 0` will re-fetch the current row. When the current row has been deleted, the next row is fetched from the cursor (or `SQLE_NOTFOUND` is returned if there are no more rows).

When a temporary table is used for the cursor, inserted rows in the underlying tables do not appear at all until that cursor is closed and reopened. It can be difficult to detect whether a temporary table is involved in a `SELECT` statement without examining the code generated by the SQL preprocessor or by becoming knowledgeable about the conditions under which temporary tables are used. Temporary tables can usually be avoided by having an index on the columns used in the `ORDER BY` clause.

Inserts, updates, and deletes to non-temporary tables may affect the cursor positioning. Because UltraLite materializes cursor rows one at a time (when temporary tables are not used), the data from a freshly inserted row (or the absence of data from a freshly deleted row) may affect subsequent `FETCH` operations. In the simple case where (parts of) rows are being selected from a single table, an inserted or updated row will appear in the result set for the cursor when it satisfies the selection criteria of the `SELECT` statement. Similarly, a freshly deleted row that previously contributed to the result set will no longer be within it.

## 3.2.2.7 User Authentication

User authentication can be controlled using the `ULGrantConnectTo` and `ULRevokeConnectFrom` methods.

The code below illustrates how to control user authentication:

```
//Embedded SQL
```

```

app() {
    ...
    /* Declare fields */
    EXEC SQL BEGIN DECLARE SECTION;
        char uid[31];
        char pwd[31];
    EXEC SQL END DECLARE SECTION;
    db_init( &sqlca );
    ...
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
    if( SQLCODE == SQLE_NOERROR ) {
        printf("Enter new user ID and password\n" );
        scanf( "%s %s", uid, pwd );
        ULGrantConnectTo( &sqlca,
            UL_TEXT( uid ), UL_TEXT( pwd ) );
        if( SQLCODE == SQLE_NOERROR ) {
            // new user added: remove DBA
            ULRevokeConnectFrom( &sqlca, UL_TEXT("DBA") );
        }
        EXEC SQL DISCONNECT;
    }
    // Prompt for password
    printf("Enter user ID and password\n" );
    scanf( "%s %s", uid, pwd );
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;
}

```

The code carries out the following tasks:

1. Initiate database functionality by calling db\_init.
2. Attempt to connect using the default user ID and password.
3. If the connection attempt is successful, add a new user.
4. If the new user is successfully added, delete the DBA user from the UltraLite database.
5. Disconnect. An updated user ID and password is now added to the database.
6. Connect using the updated user ID and password.

## 3.2.2.8 Data Encryption with UltraLite Embedded SQL

You can encrypt or obfuscate your UltraLite database using UltraLite Embedded SQL.

### Encryption

When an UltraLite database is created (using SQL Central for example), an optional encryption key may be specified. The encryption key is used to encrypt the database. Once the database is encrypted, all subsequent connection attempts must supply the encryption key. The supplied key is checked against the original encryption key and the connection fails unless the key matches.

Choose an encryption key value that cannot easily be guessed. The key can be of arbitrary length, but generally a longer key is better, because a shorter key is easier to guess. Including a combination of numbers, letters, and special characters decreases the chances of someone guessing the key.

Do not include semicolons in your key. Do not put the key itself in quotes, otherwise the quotes are considered part of the key.

The following procedure is generally used to connect to an encrypted UltraLite database:

1. Specify the encryption key in the connection string used in the EXEC SQL CONNECT statement.
2. The encryption key is specified with the key= connection string parameter.  
You must supply this key each time you want to connect to the database. Lost or forgotten keys result in completely inaccessible databases.
3. Handle attempts to open an encrypted database with the wrong key.  
If an attempt is made to open an encrypted database and the wrong key is supplied, db\_init returns ul\_false and SQLCODE -840 is set.

## Change the Encryption Key

You can change the encryption key for a database. The application must already be connected to the database using the existing key before the change can be made.

Supply the new key as an argument of the ULChangeEncryptionKey method.

## Obfuscation

Obfuscation is an option for encoding the database that is an alternative to database encryption. Obfuscation is a simple masking of the data in the database that is intended to prevent browsing the data in the database with a low level file examination utility. However, obfuscation is not secure against skilled and determined attempts to gain access to the data. Obfuscation is a database creation option and must be specified when the database is created.

### 3.2.2.9 Synchronization Setup for an Embedded SQL Application

Synchronization is a key feature of many UltraLite applications. Members of the structures in the Embedded SQL API are similar to the UltraLite C++ API.

The synchronization logic that keeps UltraLite applications up to date with the consolidated database is not held in the application itself. Synchronization scripts stored in the consolidated database, together with the MobiLink server and the UltraLite runtime library, control how changes are processed when they are uploaded and determines which changes are to be downloaded.

The specifics of each synchronization are controlled by a set of synchronization parameters. These parameters are gathered into a structure, which is then supplied as an argument in a method call to synchronize. The outline of the method is the same in each development model.

The following procedure is generally used to add synchronization to your application:

1. Initialize the structure that holds the synchronization parameters.
2. Assign the parameter values for your application.
3. Call the synchronization method, supplying the structure or object as argument.

Ensure that there are no uncommitted changes when you synchronize.

#### In this section:

##### [Synchronization Parameter Initialization \[page 700\]](#)

The synchronization parameters are stored in a `ul_sync_info` structure.

##### [Synchronization Invocation \[page 701\]](#)

The details of how to invoke synchronization depends on your target platform and on the synchronization stream.

##### [Committed Changes and Synchronization \[page 701\]](#)

An UltraLite database cannot have uncommitted changes when it is synchronized.

##### [Initial Data for Your Application \[page 701\]](#)

Many UltraLite applications need data to start working. You can download data into your application by synchronizing.

##### [Synchronization Communications Errors \[page 702\]](#)

SQLCODE can be used to report MobiLink synchronization errors.

##### [Synchronization Monitoring and Canceling \[page 702\]](#)

You can monitor and cancel synchronization from UltraLite applications.

## 3.2.2.9.1 Synchronization Parameter Initialization

The synchronization parameters are stored in a `ul_sync_info` structure.

The members of the `ul_sync_info` structure are undefined on initialization. You must set your parameters to their initial values with a call to a special method. The synchronization parameters are defined in a structure declared in the UltraLite header file `%SQLANY17%\SDK\Include\ulglobal.h`.

### Example

The following example illustrates how to initialize synchronization parameters with the `ULInitSyncInfo` method:

```
ul_sync_info synch_info;  
ULInitSyncInfo( &synch_info );
```

### Example

The following code initiates TCP/IP synchronization. The MobiLink user name is `Betty Best`, with password `TwentyFour`, the script version is `default`, and the MobiLink server is running on the host computer `test.internal`, on port 2439:

```
ul_sync_info synch_info;  
ULInitSyncInfo( &synch_info );
```

```
synch_info.user_name = UL_TEXT("Betty Best");
synch_info.password = UL_TEXT("TwentyFour");
synch_info.version = UL_TEXT("default");
synch_info.stream = ULStream();
synch_info.stream_parms =
    UL_TEXT("host=test.internal;port=2439");
ULSynchronize( &sqlca, &synch_info );
```

### 3.2.2.9.2 Synchronization Invocation

The details of how to invoke synchronization depends on your target platform and on the synchronization stream.

The synchronization process can only work if the device running the UltraLite application is able to communicate with the MobiLink server. For some platforms, the device needs to be physically connected by placing it in its cradle or by attaching it to a server computer using the appropriate cable. If the synchronization cannot be completed, add error handling code to your application.

To invoke synchronization, call the `ULInitSynchInfo` method to initialize the synchronization parameters, and then call the `ULSynchronize` method to synchronize.

The synchronization call requires a structure that holds a set of parameters describing the specifics of the synchronization. The particular parameters used depend on the stream.

### 3.2.2.9.3 Committed Changes and Synchronization

An UltraLite database cannot have uncommitted changes when it is synchronized.

If you attempt to synchronize an UltraLite database when any connection has an uncommitted transaction, the synchronization fails, an exception is thrown and the `SQLE_UNCOMMITTED_TRANSACTIONS` error is set. This error code also appears in the MobiLink server log.

### 3.2.2.9.4 Initial Data for Your Application

Many UltraLite applications need data to start working. You can download data into your application by synchronizing.

Add logic to your application to ensure that, the first time it is run, it downloads all necessary data before any other actions are carried out.

#### **i** Note

It is easier to locate errors if you develop an application in stages. When developing a prototype, temporarily use `INSERT` statements in your application to provide data for testing and demonstration purposes. Once your prototype is working correctly, replace the temporary `INSERT` statements with the code to perform the synchronization.

### 3.2.2.9.5 Synchronization Communications Errors

SQLCODE can be used to report MobilLink synchronization errors.

The following code illustrates how to handle communications errors from Embedded SQL applications:

```
if( psqlca->sqlcode == SQLE_MOBILINK_COMMUNICATIONS_ERROR ) {
    printf( " Stream error information:\n"
           "   stream_error_code = %ld\t(ss_error_code)\n"
           "   error_string      = \"%s\"\n"
           "   system_error_code = %ld\n",
           (long)info.stream_error.stream_error_code,
           info.stream_error.error_string,
           (long)info.stream_error.system_error_code );
}
```

SQLE\_MOBILINK\_COMMUNICATIONS\_ERROR is the general error code for communications errors.

To keep UltraLite small, the runtime reports numbers rather than messages.

### 3.2.2.9.6 Synchronization Monitoring and Canceling

You can monitor and cancel synchronization from UltraLite applications.

#### Monitoring Synchronization

- Specify the name of your callback function in the observer member of the synchronization structure (ul\_sync\_info).
- Call the synchronization function or method to start synchronization.
- UltraLite calls your callback function whenever the synchronization state changes.

The following code shows how this sequence of tasks can be implemented in an Embedded SQL application:

```
ULInitSyncInfo( &info );
info.user_name = m_EmpIDStr;
...
//The info parameter of ULSynchronize() contains
// a pointer to the observer function
info.observer = ObserverFunc;
ULSynchronize( &sqlca, &info );
```

**In this section:**

[Synchronization Status Information \[page 703\]](#)

The callback function that monitors synchronization takes a ul\_sync\_status structure as a parameter.

### 3.2.2.9.6.1 Synchronization Status Information

The callback function that monitors synchronization takes a `ul_sync_status` structure as a parameter.

**sent.inserts**

The number of inserted rows that have been uploaded so far.

**sent.updates**

The number of updated rows that have been uploaded so far.

**sent.deletes**

The number of deleted rows that have been uploaded so far.

**sent.bytes**

The number of bytes that have been uploaded so far.

**received.inserts**

The number of inserted rows that have been downloaded so far.

**received.updates**

The number of updated rows that have been downloaded so far.

**received.deletes**

The number of deleted rows that have been downloaded so far.

**received.bytes**

The number of bytes that have been downloaded so far.

**info**

Returns a pointer to the `ul_sync_info` structure.

**db\_table\_count**

Returns the number of tables in the database.

**table\_id**

Returns the current table number (relative to 1) that is being uploaded or downloaded. This number may skip values when not all tables are being synchronized and is not necessarily increasing.

**table\_name[]**

Returns the name of the current table.

**table\_name\_w2[]**

Returns the name of the current table (wide character version). This field is only populated in the Windows (desktop and Mobile) environment.

**sync\_table\_count**

Returns the number of tables being synchronized.

**sync\_table\_index**

Returns the number of the table that is being uploaded or downloaded, starting at 1 and ending at the `sync_table_count` value. This number may skip values when not all tables are being synchronized.

**state**

One of the following states:

**UL\_SYNC\_STATE\_STARTING**

No synchronization actions have been taken.

**UL\_SYNC\_STATE\_CONNECTING**

The synchronization stream has been built, but not opened.

**UL\_SYNC\_STATE\_RESUMING\_DOWNLOAD**

An optional state denoting an attempt to resume a partial download. When successful, the synchronization proceeds to the UL\_SYNC\_STATE\_RECEIVING\_TABLE state.

**UL\_SYNC\_STATE\_SENDING\_HEADER**

The synchronization stream has been opened, and the header is about to be sent.

**UL\_SYNC\_STATE\_SENDING\_CHECK\_SYNC\_REQUEST**

The state of the last upload is unknown, so a request to check its status is being sent.

**UL\_SYNC\_STATE\_WAITING\_CHECK\_SYNC\_REQUEST**

The client is waiting for the server to respond to the check synchronization request.

**UL\_SYNC\_STATE\_PROCESSING\_CHECK\_SYNC\_REQUEST**

The response to the check synchronization request has been received and is being processed.

**UL\_SYNC\_STATE\_SENDING\_TABLE**

A table is being sent.

**UL\_SYNC\_STATE\_SENDING\_DATA**

Schema information or data is being sent.

**UL\_SYNC\_STATE\_FINISHING\_UPLOAD**

The upload stage has completed and a commit is being carried out.

**UL\_SYNC\_STATE\_WAITING\_UPLOAD\_ACK**

The client is waiting for the server to acknowledge receiving the upload.

**UL\_SYNC\_STATE\_PROCESSING\_UPLOAD\_ACK**

The server has acknowledged receiving the upload.

**UL\_SYNC\_STATE\_WAITING\_FOR\_DOWNLOAD**

The client is waiting for the server to start sending the download.

**UL\_SYNC\_STATE\_RECEIVING\_TABLE**

A table is being received.

**UL\_SYNC\_STATE\_RECEIVING\_DATA**

Schema information or data is being received.

**UL\_SYNC\_STATE\_COMMITTING\_DOWNLOAD**

The download stage is completed and a commit is being carried out.

**UL\_SYNC\_STATE\_SENDING\_DOWNLOAD\_ACK**

An acknowledgement that the download is complete is being sent.

**UL\_SYNC\_STATE\_DISCONNECTING**

The synchronization stream is about to be closed.

**UL\_SYNC\_STATE\_DONE**

Synchronization has completed successfully.



## **UL\_SYNC\_STATE\_ERROR**

Synchronization has completed, but with an error.

## **UL\_SYNC\_STATE\_ROLLING\_BACK\_DOWNLOAD**

An error occurred during download, and the download is being rolled back.

### **stop**

Set this member to true to interrupt the synchronization. The SQL exception `SQLE_INTERRUPTED` is set, and the synchronization stops as if a communications error had occurred. The observer is *always* called with either the `DONE` or `ERROR` state so that it can do proper cleanup.

### **flags**

Returns the current synchronization flags indicating additional information related to the current state.

### **user\_data**

Returns the user data object that is passed as an argument to the `ULSetSynchronizationCallback` function.

### **sqlca**

Returns the pointer to the connection's active SQLCA.

### **current\_download\_row\_count**

Returns the number of rows that have been downloaded so far. This number includes duplicate rows that aren't included in `received.inserts`, `received.updates`, or `received.deletes`.

### **total\_download\_row\_count**

Returns the total number of rows to be received in the download. This number includes duplicate rows that aren't included in `received.inserts`, `received.updates`, or `received.deletes`.

## **Example**

The following code illustrates a simple observer function:

```
extern void __stdcall ObserverFunc(
    p_ul_sync_status status )
{
    switch( status->state ) {
        case UL_SYNC_STATE_STARTING:
            printf( "Starting\n" );
            break;
        case UL_SYNC_STATE_CONNECTING:
            printf( "Connecting\n" );
            break;
        case UL_SYNC_STATE_SENDING_HEADER:
            printf( "Sending Header\n" );
            break;
        case UL_SYNC_STATE_SENDING_TABLE:
            printf( "Sending Table %d of %d\n",
                status->tableIndex + 1,
                status->tableCount );
            break;
        case UL_SYNC_STATE_RECEIVING_TABLE:
            printf( "Receiving Table %d of %d\n",
                status->tableIndex + 1,
                status->tableCount );
            break;
        case UL_SYNC_STATE_SENDING_DOWNLOAD_ACK:
            printf( "Sending Download Ack\n" );
            break;
    }
}
```

```

        break;
    case UL_SYNC_STATE_DISCONNECTING:
        printf( "Disconnecting\n" );
        break;
    case UL_SYNC_STATE_DONE:
        printf( "Done\n" );
        break;
    ...

```

This observer produces the following output when you synchronize two tables:

```

Starting
Connecting
Sending Header
Sending Table 1 of 2
Sending Table 2 of 2
Receiving Upload Ack
Receiving Table 1 of 2
Receiving Table 2 of 2
Sending Download Ack
Disconnecting
Done

```

## CustDB example

An example of an observer function is included in the CustDB sample application. The implementation in CustDB provides a window that displays synchronization progress and allows the user to cancel synchronization. The user-interface component makes the observer function platform specific.

The CustDB sample code is in the `%SQLANYSAMP17%\UltraLite\CustDB` directory. The observer function is contained in platform-specific subdirectories of the CustDB directory.

### 3.2.2.10 Embedded SQL Application Building

Building an Embedded SQL application requires knowledge of several configuration settings.

#### In this section:

##### [General Build Procedures \[page 707\]](#)

Building an UltraLite Embedded SQL application consists of running the SQL preprocessor, compiling source code, and then linking to object files.

##### [Development Tool Configuration for Embedded SQL Development \[page 707\]](#)

Many development tools use a dependency model, sometimes expressed as a makefile, in which the timestamp on each source file is compared with that on the target file (usually the object file) to decide whether the target file needs to be regenerated.

## 3.2.2.10.1 General Build Procedures

Building an UltraLite Embedded SQL application consists of running the SQL preprocessor, compiling source code, and then linking to object files.

1. Run the SQL preprocessor on *each* Embedded SQL source file.  
The SQL preprocessor is the `sqlpp` command line utility. It preprocesses the Embedded SQL source files, producing C++ source files to be compiled into your application.

### ⚠ Caution

`sqlpp` overwrites the output file without regard to its contents. Ensure that the output file name does not match the name of any of your source files. By default, `sqlpp` constructs the output file name by changing the suffix of your source file to `.cpp`. When in doubt, specify the output file name explicitly, following the name of the source file.

2. Compile *each* C++ source file for the target platform of your choice. Include:
  - each C++ file generated by the SQL preprocessor
  - any additional C or C++ source files required by your application
3. Link *all* these object files, together with the UltraLite runtime library.

## 3.2.2.10.2 Development Tool Configuration for Embedded SQL Development

Many development tools use a dependency model, sometimes expressed as a makefile, in which the timestamp on each source file is compared with that on the target file (usually the object file) to decide whether the target file needs to be regenerated.

With UltraLite development, a change to any SQL statement in a development project means that the generated code needs to be regenerated. Changes are not reflected in the timestamp on any individual source file because the SQL statements are stored in the reference database.

### In this section:

#### [Running the SQL Preprocessor \[page 708\]](#)

Incorporate the SQL preprocessor into a dependency-based build environment by adding instructions to run it for Microsoft Visual C++.

## 3.2.2.10.2.1 Running the SQL Preprocessor

Incorporate the SQL preprocessor into a dependency-based build environment by adding instructions to run the it for Microsoft Visual C++.

### Procedure

1. Add the `.sqc` files to your development project.

The development project is defined in your development tool.

2. Add a custom build rule for each `.sqc` file.

- The custom build rule should run the SQL preprocessor. In Microsoft Visual C++, the build rule should have the following command (entered on a single line):

```
"%SQLANY17%\Bin32\sqlpp.exe" -q -u $(InputPath) $(InputName).cpp
```

where `SQLANY17` is an environment variable that points to your SQL Anywhere installation directory.

- Set the output for the command to `$(InputName).cpp`.
3. Compile the `.sqc` files, and add the generated `.cpp` files to your development project.

You must add the generated files to your project even though they are not source files, so that you can set up dependencies and build options.

4. For each generated `.cpp` file, set the preprocessor definitions.
  - Under General or Preprocessor, add `UL_USE_DLL` to the Preprocessor definitions.
  - Under Preprocessor, add `$(SQLANY17)\SDK\Include` and any other desired include folders to your include path as a comma-separated list.

### Results

The SQL preprocessor is configured for Microsoft Visual C++ development.

## 3.2.3 UltraLite Application Development for Microsoft Windows Mobile

Microsoft Visual Studio 2005 and later can be used to develop applications for the Microsoft Windows Mobile environment.

Applications targeting Microsoft Windows Mobile should use the default setting for `wchar_t` and link against the UltraLite runtime libraries in `\Program Files\SQLAny17\ultralite\ce\arm.50\lib\`.

You can test your applications under an emulator on most Microsoft Windows Mobile target platforms.

#### In this section:

##### [CustDB Sample Application \[page 709\]](#)

CustDB is a simple sales-status application that is provided as a Microsoft Visual Studio solution. It is located in the `%SQLANYSAMPI7%\UltraLite\` directory of your SQL Anywhere installation.

##### [Persistent Data \[page 710\]](#)

The UltraLite database is stored in the Microsoft Windows Mobile file system.

##### [Assigning Class Names for Applications \[page 711\]](#)

Assign a distinct class name for your application if you are using MFC. When registering applications for use with ActiveSync you must supply a window class name.

##### [Microsoft Windows Mobile Synchronization \[page 712\]](#)

UltraLite applications on Microsoft Windows Mobile can synchronize through the Microsoft ActiveSync, TCP/IP, and HTTP stream types.

### 3.2.3.1 CustDB Sample Application

CustDB is a simple sales-status application that is provided as a Microsoft Visual Studio solution. It is located in the `%SQLANYSAMPI7%\UltraLite\` directory of your SQL Anywhere installation.

#### **i** Note

The sample project uses environment variables wherever possible. It may be necessary to adjust the project for the application to build properly. If you experience problems, try searching for missing files in the Microsoft Visual C++ folder(s) and adding the appropriate directory settings.

#### In this section:

##### [Building the CustDB Sample Application \[page 710\]](#)

Build the CustDB sample application to see how an application interfaces with an UltraLite database.

### Related Information

[Embedded SQL Application Building \[page 706\]](#)

### 3.2.3.1 Building the CustDB Sample Application

Build the CustDB sample application to see how an application interfaces with an UltraLite database.

#### Procedure

1. Start Microsoft Visual Studio.
2. Open the project file located in the `%SQLANYSAMP17%\UltraLite\CustDB` directory.
3. Click **Build > Set Configuration Manager** to set the target platform.

Set an active solution platform of your choice.

4. Build the application:
  - Click **Build > Deploy Solution** to build and deploy CustDB.

When the application is built it will be uploaded automatically to the remote device.

5. Start the MobiLink server:
  - To start the MobiLink server, click **Start > Programs > SQL Anywhere 17 > MobiLink > Synchronization Server Sample**.

6. Run the CustDB application:

Before running the CustDB application, the `custdb` database must be copied to the root folder of the device. Copy the database file named `%SQLANYSAMP17%\UltraLite\CustDB\custdb.udb` to the root of the device.

On the device or simulator, execute `CustDB.exe`, which is located in the project folder under `\Program Files`.

#### Results

The CustDB application loads.

### 3.2.3.2 Persistent Data

The UltraLite database is stored in the Microsoft Windows Mobile file system.

The default file is `\UltraLiteDB\ul_store.udb`. You can override this choice using the `file_name` connection parameter which specifies the full path name of the file-based persistent store.

The UltraLite runtime carries out no substitutions on the `file_name` parameter. If a directory has to be created for the file name to be valid, the application must ensure that any directories are created before calling `db_init`.

As an example, you could make use of a flash memory storage card by scanning for storage cards and prefixing a name by the appropriate directory name for the storage card. For example:

```
file_name = "\\Storage Card\\My Documents\\flash.udb"
```

### 3.2.3.3 Assigning Class Names for Applications

Assign a distinct class name for your application if you are using MFC. When registering applications for use with ActiveSync you must supply a window class name.

#### Prerequisites

Assigning class names is carried out at development time and your application development tool documentation is the primary source of information about the topic.

#### Procedure

1. Create and register a custom window class for dialog boxes, based on the default class.

Add the following code to your application's startup code. The code must be executed before any dialogs get created.

```
WNDCLASS wc;
if( ! GetClassInfo( NULL, L"Dialog", &wc ) ) {
    AfxMessageBox( L"Error getting class info" );
}
wc.lpszClassName = L"MY_APP_CLASS";
if( ! AfxRegisterClass( &wc ) ) {
    AfxMessageBox( L"Error registering class" );
}
```

where `MY_APP_CLASS` is the unique class name for your application.

2. Determine which dialog is the main dialog for your application.

If your project was created with the MFC Application Wizard, this is likely to be a dialog named `MyAppDlg`.

3. Find and record the resource ID for the main dialog.

The resource ID is a constant of the same general form as `IDD_MYAPP_DIALOG`.

4. Ensure that the main dialog remains open any time your application is running.

Add the following code to your application's `InitInstance` method.

```
m_pMainWnd = &dlg;
```

The code ensures that if the `dlg` main dialog is closed, the application also closes. For more information, see the Microsoft documentation for `CWinThread::m_pMainWnd`.

If the dialog does not remain open for the duration of your application, you must change the window class of other dialogs as well.

5. Save your changes.

6. Modify the resource file for your project.

Open your resource file (which has an extension of `.rc`) in a text editor such as Notepad.

7. Locate the resource ID of your main dialog.

Change the main dialog's definition to use the new window class as in the following example. The *only* change that you should make is the addition of the CLASS line:

```
IDD_MYAPP_DIALOG_DIALOG DISCARDABLE 0, 0, 139, 103
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION
EXSTYLE WS_EX_APPWINDOW | WS_EX_CAPTIONOKBTN
CAPTION "MyApp"
FONT 8, "System"
CLASS "MY_APP_CLASS"
BEGIN
    LTEXT "TODO: Place dialog controls here.", IDC_STATIC, 13, 33, 112, 17
END
```

where `MY_APP_CLASS` is the name of the window class you used earlier.

8. Save the `.rc` file.

9. Add code to catch the synchronization message.

## Results

A distinct class name for your application is created.

## Related Information

[Adding ActiveSync Synchronization in the Main Dialog Class \[page 714\]](#)

### 3.2.3.4 Microsoft Windows Mobile Synchronization

UltraLite applications on Microsoft Windows Mobile can synchronize through the Microsoft ActiveSync, TCP/IP, and HTTP stream types.

The `user_name` and `stream_parms` parameters must be surrounded by the `UL_TEXT()` macro for Microsoft Windows Mobile when initializing, since the compilation environment is Unicode wide characters.

**In this section:**

[Microsoft ActiveSync Synchronization in UltraLite C++ \[page 713\]](#)



Microsoft ActiveSync handles data synchronization between a desktop computer running Microsoft Windows and a connected Microsoft Windows Mobile handheld device. UltraLite supports Microsoft ActiveSync versions 3.5 and later.

#### [Microsoft ActiveSync Synchronization in UltraLite C++ \(Windows API\) \[page 714\]](#)

If you are programming directly to the Windows API, you must handle the message from the MobiLink provider in your application's WindowProc function, using the ULLsSynchronizeMessage function to determine if it has received the message.

#### [Adding ActiveSync Synchronization in the Main Dialog Class \[page 714\]](#)

Catch synchronization messages in your main dialog class.

#### [Adding ActiveSync Synchronization in the Application Class \[page 716\]](#)

Catch synchronization messages in your application class.

#### [TCP/IP, HTTP, or HTTPS Synchronization from Microsoft Windows Mobile \[page 717\]](#)

For TCP/IP, HTTP, or HTTPS synchronization, the application controls when synchronization occurs. Your application should provide a menu item or user interface control so that the user can request synchronization.

### **3.2.3.4.1 Microsoft ActiveSync Synchronization in UltraLite C++**

Microsoft ActiveSync handles data synchronization between a desktop computer running Microsoft Windows and a connected Microsoft Windows Mobile handheld device. UltraLite supports Microsoft ActiveSync versions 3.5 and later.

If you use Microsoft ActiveSync, synchronization can be initiated only by Microsoft ActiveSync itself. Microsoft ActiveSync automatically initiates a synchronization when the device is placed in the cradle or when *Synchronize* is selected from the Microsoft ActiveSync window. The MobiLink provider starts the application, if it is not already running, and sends a message to the application.

The Microsoft ActiveSync provider uses the wParam parameter. A wParam value of 1 indicates that the MobiLink provider for Microsoft ActiveSync launched the application. The application must then shut itself down after it has finished synchronizing. If the application was already running when called by the MobiLink provider for Microsoft ActiveSync, wParam is 0. The application can ignore the wParam parameter if it wants to keep running.

Adding synchronization depends on whether you are addressing the Microsoft Windows API directly or whether you are using the Microsoft Foundation Classes. Both development models are described here.

### 3.2.3.4.2 Microsoft ActiveSync Synchronization in UltraLite C++ (Windows API)

If you are programming directly to the Windows API, you must handle the message from the MobiLink provider in your application's WindowProc function, using the ULIIsSynchronizeMessage function to determine if it has received the message.

Here is an example of how to handle the message:

```
LRESULT CALLBACK WindowProc( HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam )
{
    if( ULIIsSynchronizeMessage( uMsg ) ) {
        DoSync();
        if( wParam == 1 ) DestroyWindow( hwnd );
        return 0;
    }
    switch( uMsg ) {
        // code to handle other windows messages
        default:
            return DefWindowProc( hwnd, uMsg, wParam, lParam );
    }
    return 0;
}
```

where DoSync is the method that actually calls ULSynchronize.

### 3.2.3.4.3 Adding ActiveSync Synchronization in the Main Dialog Class

Catch synchronization messages in your main dialog class.

#### Prerequisites

You must use Microsoft Foundation Classes to develop your application

#### Context

Your application must create and register a custom window class name for notification.

## Procedure

1. Add a registered message and declare a message handler.

Find the message map in the source file for your main dialog (the name is of the same form as `CMyAppDlg.cpp`). Add a registered message using the *static* and declare a message handler using `ON_REGISTERED_MESSAGE` as in the following example:

```
static UINT WM_ULTRALITE_SYNC_MESSAGE =
    ::RegisterWindowMessage( UL_AS_SYNCHRONIZE );
BEGIN_MESSAGE_MAP(CMyAppDlg, CDialog)
   //{{AFX_MSG_MAP(CMyAppDlg)
    //}}AFX_MSG_MAP
    ON_REGISTERED_MESSAGE( WM_ULTRALITE_SYNC_MESSAGE,
        OnDoUltraLiteSync )
END_MESSAGE_MAP()
```

2. Implement the message handler.

Add a method to the main dialog class with the following signature. This method is automatically executed any time the MobiLink provider for ActiveSync requests that your application synchronize. The method should call the `ULSynchronize` method.

```
LRESULT CMyAppDlg::OnDoUltraLiteSync (
    WPARAM wParam,
    LPARAM lParam
);
```

The return value of this function should be 0.

## Results

The main dialog class performs a synchronization.

## Related Information

[Assigning Class Names for Applications \[page 711\]](#)

## 3.2.3.4.4 Adding ActiveSync Synchronization in the Application Class

Catch synchronization messages in your application class.

### Prerequisites

You must use Microsoft Foundation Classes to develop your application.

Your application must create and register a custom window class name for notification.

### Procedure

1. Open the *Class Wizard* for the application class.
2. In the *Messages* list, highlight *PreTranslateMessage* and then click *Add Function*.
3. Click *Edit Code*.

The PreTranslateMessage method appears.

4. Change the PreTranslateMessage method to read as follows:

```
BOOL CMyApp::PreTranslateMessage(MSG* pMsg)
{
    if( ULIsSynchronizeMessage(pMsg->message) ) {
        DoSync();
        // close application if launched by provider
        if( pMsg->wParam == 1 ) {
            ASSERT( AfxGetMainWnd() != NULL );
            AfxGetMainWnd()->SendMessage( WM_CLOSE );
        }
        return TRUE; // message has been processed
    }
    return CWinApp::PreTranslateMessage(pMsg);
}
```

where the DoSync method calls the ULSynchronize method.

### Results

The PreTranslateMessage method performs a synchronization.

### Related Information

[Assigning Class Names for Applications \[page 711\]](#)

### 3.2.3.4.5 TCP/IP, HTTP, or HTTPS Synchronization from Microsoft Windows Mobile

For TCP/IP, HTTP, or HTTPS synchronization, the application controls when synchronization occurs. Your application should provide a menu item or user interface control so that the user can request synchronization.

## 3.3 Tutorial: Building a Windows Application using the C++ API

This tutorial guides you through the process of building an UltraLite C++ application. The application is built for Windows desktop operating systems, and runs at a command prompt.

### Prerequisites

This tutorial assumes:

- You are familiar with the C++ programming language
- You have a C++ compiler installed on your computer
- You know how to create an UltraLite database with the [Create Database Wizard](#).

### Context

The goal for the tutorial is to gain competence with the process of developing an UltraLite C++ application.

This tutorial is based on development using Microsoft Visual C++, although you can also use any C++ development environment.

The tutorial takes about 30 minutes if you copy and paste the code. The final section of this tutorial contains the full source code of the program described in this tutorial.

1. [Lesson 1: Creating and Connecting to a Database \[page 718\]](#)  
In this lesson, you create an UltraLite database. You then write, compile, and run a C++ application that accesses the database you created.
2. [Lesson 2: Inserting Data into the Database \[page 721\]](#)  
In this lesson, you add data to a database.
3. [Lesson 3: Selecting and Listing Rows from the Table \[page 723\]](#)  
In this lesson, you retrieve rows from the table and print them on the command line.
4. [Lesson 4: Adding Synchronization to Your Application \[page 725\]](#)  
In this lesson, you add synchronization code to your application, start the MobiLink server, and run your application to synchronize with the consolidated database.

5. [Reviewing the Code Listing for the Tutorial \[page 726\]](#)

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

## 3.3.1 Lesson 1: Creating and Connecting to a Database

In this lesson, you create an UltraLite database. You then write, compile, and run a C++ application that accesses the database you created.

### Prerequisites

This lesson assumes that you have installed the required software.

### Procedure

1. Set the VCINSTALLDIR environment variable to the root directory of your Microsoft Visual C++ installation if the variable does not already exist.
2. Add %VCINSTALLDIR%\VC\atlmfc\src\atl to your INCLUDE environment variable.
3. Create a directory to contain the files you will create in this tutorial.

The remainder of this tutorial assumes that this directory is C:\tutorial\cpp\. If you create a directory with a different name, use that directory instead of C:\tutorial\cpp\.

4. Using UltraLite in SQL Central, create a database named `ULCustomer.udb` in your new directory with the default characteristics.
5. Add a table named `ULCustomer` to the database. Use the following specifications for the `ULCustomer` table:

Column name	Data type (size)	Columns allows NULL values?	Default value	Primary key
<code>cust_id</code>	integer	No	autoincrement	ascending
<code>cust_name</code>	varchar(30)	No	None	

6. Disconnect from the database in SQL Central, otherwise your executable will not be able to connect.
7. In Microsoft Visual C++, click **File > New**.
8. On the *Files* tab, click *C++ Source File*.
9. Save the file as `customer.cpp` in your tutorial directory.
10. Include the UltraLite libraries.

Copy the code below into `customer.cpp`:

```
#include <tchar.h>
#include <stdio.h>
#include "ulcpp.h"
```

```
#define MAX_NAME_LEN 100
#define MAX_ERROR_LEN 256
```

11. Define connection parameters to connect to the database.

In this code fragment, the connection parameters are hard coded. In a real application, the locations might be specified at runtime.

Copy the code below into `customer.cpp`.

```
static ul_char const * ConnectionParms =
    "UID=DBA;PWD=sql;DBF=C:\\tutorial\\cpp\\ULCustomer.udb";
```

### Note

A backslash character that appears in the file name location string must be escaped by a preceding backslash character.

12. Define a method for handling database errors in the application.

UltraLite provides a callback mechanism to notify the application of errors. In a development environment this method can be useful as a mechanism to handle errors that were not anticipated. A production application typically includes code to handle all common error situations. An application can check for errors after every call to an UltraLite method or can choose to use an error callback function.

The following code is a sample callback function:

```
ul_error_action UL_CALLBACK_FN MyErrorCallBack(
    const ULError * error,
    ul_void * user_data )
{
    ul_error_action rc;
    an_sql_code code = error->GetSQLCode();

    (void) user_data;
    switch( code ){
        // The following error is used for flow control - don't report it here
        case SQLE_NOTFOUND:
            rc = UL_ERROR_ACTION_CONTINUE;
            break;
        default:
            if (code >= 0) { // warning or success
                rc = UL_ERROR_ACTION_DEFAULT;
            } else { // negative is real error
                ul_char etext[ MAX_ERROR_LEN ];
                error->GetString( etext, MAX_ERROR_LEN );
                _tprintf( "Error %ld: %s\n", code, etext );
                rc = UL_ERROR_ACTION_CANCEL;
            }
            break;
    }
    return rc;
}
```

In UltraLite, the error `SQLE_NOTFOUND` is often used to control application flow. That error is signaled to mark the end of a loop over a result set. The generic error handler coded above does not output a message for this error condition.

13. Define a method to open a connection to a database.

```
static ULConnection * open_conn( void ) {
    ULConnection * conn =
    ULDatabaseManager::OpenConnection( ConnectionParms );
```

```

if( conn == UL_NULL ) {
    _tprintf("Unable to open existing database.\n");
}
return conn;
}

```

If the database file does not exist, an error message is displayed, otherwise a connection is established.

14. Implement the main method to perform the following tasks:

- Registers the error handling method.
- Opens a connection to the database.
- Closes the connection and finalizes the database manager.

```

int main() {
    ULConnection * conn;
    ULDatabaseManager::Init();
    ULDatabaseManager::SetErrorCallback( MyErrorCallBack, NULL );
    conn = open_conn();
    if ( conn == UL_NULL ) {
        ULDatabaseManager::Fini();
        return 1;
    }
    // Main processing code goes here ...
    conn->Close();
    ULDatabaseManager::Fini();
    return 0;
}

```

15. Compile and link the source file.

The method you use to compile the source file depends on your compiler. The following instructions are for the Microsoft Visual C++ command line compiler using a makefile:

- Open a command prompt and change to your tutorial directory.
- Create a makefile named `makefile`.
- In the makefile, add directories to your include path.

```
IncludeFolders=/I"${SQLANY17}\SDK\Include"
```

- In the makefile, add directories to your libraries path.

```
LibraryFolders=/LIBPATH:"${SQLANY17}\UltraLite\Windows\x86\Lib\vs9"
```

- In the makefile, add libraries to your linker options.

```
Libraries=ulimp.lib
```

The UltraLite runtime library is named `ulimp.lib`.

- In the makefile, set compiler options. You must enter these options on a single line.

```
CompileOptions=/c /nologo /W3 /Od /Zi /DWIN32 /DUL_USE_DLL
```

- In the makefile, add an instruction for linking the application.

```
customer.exe: customer.obj
    link /NOLOGO /DEBUG customer.obj $(LibraryFolders) $(Libraries)
```

- In the makefile, add an instruction for compiling the application.

```
customer.obj: customer.cpp
```



```
cl $(CompileOptions) $(IncludeFolders) customer.cpp
```

- i. Run `vsvars32.bat`.

```
%VCINSTALLDIR%\Tools\vsvars32.bat
```

- j. Run the makefile.

```
nmake
```

This creates an executable named `customer.exe`.

16. Run the application.

At a command prompt, enter `customer`.

## Results

The application connects to the database and then disconnects. The application runs successfully when you do not see any error messages.

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building a Windows Application using the C++ API \[page 717\]](#)

**Next task:** [Lesson 2: Inserting Data into the Database \[page 721\]](#)

## Related Information

[Error Handling \[page 665\]](#)

## 3.3.2 Lesson 2: Inserting Data into the Database

In this lesson, you add data to a database.

## Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

## Procedure

1. Add the method below to `customer.cpp` immediately before the main method:

```
static bool do_insert( ULConnection * conn )
{
    ULTable * table = conn->OpenTable( "ULCustomer" );
    if( table == UL_NULL ) {
        _tprintf( "Table not found: ULCustomer\n" );
        return false;
    }
    if( table->GetRowCount() == 0 ) {
        _tprintf( "Inserting one row.\n" );
        table->InsertBegin();
        table->SetString( "cust_name", "New Customer" );
        table->Insert();
        conn->Commit();
    } else {
        _tprintf( "The table has %lu rows\n", table->GetRowCount() );
    }
    table->Close();
    return true;
}
```

This method performs the following tasks.

- Opens the table using the `connection->OpenTable()` method. You must open a Table object to perform operations on the table.
  - If the table is empty, adds a row to the table. To insert a row, the code changes to insert mode using the `InsertBegin` method, sets values for each required column, and executes an insert to add the row to the database.
  - If the table is not empty, reports the number of rows in the table.
  - Closes the Table object to free resources associated with it.
  - Returns a boolean indicating whether the operation was successful.
2. Call the `do_insert` method you have created.

Add the following line to the `main()` method, immediately before the call to `conn->Close`.

```
do_insert(conn);
```

3. Compile your application by running `nmake`.
4. Run your application by typing `customer` at a command prompt.

## Results

The application runs and you can insert data into the `ULCustomer` table.

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building a Windows Application using the C++ API \[page 717\]](#)

**Previous task:** [Lesson 1: Creating and Connecting to a Database \[page 718\]](#)

**Next task:** [Lesson 3: Selecting and Listing Rows from the Table \[page 723\]](#)

## 3.3.3 Lesson 3: Selecting and Listing Rows from the Table

In this lesson, you retrieve rows from the table and print them on the command line.

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

### Procedure

1. Add the method below to `customer.cpp` immediately after the `do_insert` method.

```
static bool do_select( ULConnection * conn )
{
    ULTable * table = conn->OpenTable( "ULCustomer" );
    if( table == UL_NULL ) {
        return false;
    }
    ULTableSchema * schema = table->GetTableSchema();
    if( schema == UL_NULL ) {
        table->Close();
        return false;
    }
    ul_column_num id_cid =
        schema->GetColumnID( "cust_id" );
    ul_column_num cname_cid =
        schema->GetColumnID( "cust_name" );
    schema->Close();
    _tprintf( "\n\nTable 'ULCustomer' row contents:\n" );
    while( table->Next() ) {
        ul_char cname[ MAX_NAME_LEN ];
        table->GetString( cname_cid, cname, MAX_NAME_LEN );
        _tprintf( "id=%d, name=%s\n", (int)table->GetInt(id_cid), cname );
    }
    table->Close();
    return true;
}
```

```
}
```

This method carries out the following tasks:

- Opens the Table object.
- Retrieves the column identifiers.
- Sets the current position before the first row of the table.  
Any operations on a table are carried out at the current position. The position may be before the first row, on one of the rows of the table, or after the last row. By default, as in this case, the rows are ordered by their primary key value (`cust_id`). To order rows in a different way, you can add an index to an UltraLite database and open a table using that index.
- For each row, the `cust_id` and `cust_name` values are written out. The loop carries on until the `Next` method returns false, which occurs after the final row.
- Closes the Table object.

2. Add the following line to the main method immediately after the call to the `insert` method:

```
do_select(conn);
```

3. Compile your application by running `nmake`.
4. Run your application by typing `customer` at a command prompt.

## Results

A list of all the customer IDs and customer names in the `ULCustomer` table is outputted.

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building a Windows Application using the C++ API \[page 717\]](#)

**Previous task:** [Lesson 2: Inserting Data into the Database \[page 721\]](#)

**Next task:** [Lesson 4: Adding Synchronization to Your Application \[page 725\]](#)

## 3.3.4 Lesson 4: Adding Synchronization to Your Application

In this lesson, you add synchronization code to your application, start the MobiLink server, and run your application to synchronize with the consolidated database.

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

### Context

The UltraLite database you created in the previous lessons synchronizes with the UltraLite 17 Sample database. The UltraLite 17 Sample database has a ULCustomer table whose columns include those in the customer table of your local UltraLite database.

### Procedure

1. Add the method below to `customer.cpp`.

```
static bool do_sync( ULConnection * conn )
{
    ul_sync_info info;
    ul_stream_error * se = &info.stream_error;

    ULDatabaseManager::EnableTcpipSynchronization();
    conn->InitSyncInfo( &info );
    info.stream = "TCPIP";
    info.version = "custdb 17.0";
    info.user_name = "50";
    info.download_only = true;
    if( !conn->Synchronize( &info ) ) {
        _tprintf( "Synchronization error \n" );
        _tprintf( "  stream_error_code is '%lu'\n", se->stream_error_code );
        _tprintf( "  system_error_code is '%ld'\n", se->system_error_code );
        _tprintf( "  error_string is '" );
        _tprintf( "%s", se->error_string );
        _tprintf( "'\n" );
        return false;
    }
    return true;
}
```

This method carries out the following tasks:

- Enables TCP/IP communications by invoking `EnableTcpipSynchronization`. Synchronization can also be carried out over HTTP, HTTPS, and TLS.

- Sets the script version. MobiLink synchronization is controlled by scripts stored in the consolidated database. The script version identifies which set of scripts to use.
  - Sets the MobiLink user name. This value is used for authentication at the MobiLink server. It is distinct from the UltraLite database user ID, although in some applications you may want to give them the same value.
  - Sets the `download_only` parameter to true. By default, MobiLink synchronization is two-way. This application uses download-only synchronization so that the rows in your table do not get uploaded to the sample database.
2. Place the following line of code to the main method after the `do_select` method call:

```
do_sync(conn);
```

3. Compile your application by running `nmake`.
4. Start the MobiLink server.

At a command prompt, run the following command:

```
mldrsv17 -c "dsn=SQL Anywhere 17 CustDB;uid=ml_server;pwd=sql" -v -vr -vs -zu+
-o custdbASA.log
```

The `-zu+` option provides automatic addition of users. The `-v+` option turns on verbose logging for all messages.

5. Run your application by typing `customer` at a command prompt.

## Results

The MobiLink server messages window displays status messages indicating the synchronization progress. If synchronization is successful, the final message displays `Synchronization complete`.

**Task overview:** [Tutorial: Building a Windows Application using the C++ API \[page 717\]](#)

**Previous task:** [Lesson 3: Selecting and Listing Rows from the Table \[page 723\]](#)

**Next:** [Reviewing the Code Listing for the Tutorial \[page 726\]](#)

### 3.3.5 Reviewing the Code Listing for the Tutorial

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

```
#include <tchar.h>
#include <stdio.h>
#include "ulcpp.h"
#define MAX_NAME_LEN 100
#define MAX_ERROR_LEN 256
static ul_char const * ConnectionParms =
    "UID=DBA;PWD=sql;DBF=c:\\tutorial\\cpp\\ULCustomer.udb";
ul_error_action UL_CALLBACK_FN MyErrorCallback(
```

```

const ULError * error,
ul_void * user_data )
{
    ul_error_action rc;
    an_sql_code code = error->GetSQLCode();

    (void) user_data;
    switch( code ){
        // The following error is used for flow control - don't report it here
        case SQLE_NOTFOUND:
            rc = UL_ERROR_ACTION_CONTINUE;
            break;
        default:
            if (code >= 0) { // warning or success
                rc = UL_ERROR_ACTION_DEFAULT;
            } else { // negative is real error
                ul_char etext[ MAX_ERROR_LEN ];
                error->GetString( etext, MAX_ERROR_LEN );
                _tprintf( "Error %ld: %s\n", code, etext );
                rc = UL_ERROR_ACTION_CANCEL;
            }
            break;
    }
    return rc;
}

static ULConnection * open_conn( void ) {
    ULConnection * conn = ULDatabaseManager::OpenConnection( ConnectionParms );
    if( conn == UL_NULL ) {
        _tprintf( "Unable to open existing database.\n" );
    }
    return conn;
}

static bool do_insert( ULConnection * conn ) {
    ULTable * table = conn->OpenTable( "ULCustomer" );
    if( table == UL_NULL ) {
        _tprintf( "Table not found: ULCustomer\n" );
        return false;
    }
    if( table->GetRowCount() == 0 ) {
        _tprintf( "Inserting one row.\n" );
        table->InsertBegin();
        table->SetString( "cust_name", "New Customer" );
        table->Insert();
        conn->Commit();
    } else {
        _tprintf( "The table has %lu rows\n",
            table->GetRowCount() );
    }
    table->Close();
    return true;
}

static bool do_select( ULConnection * conn )
{
    ULTable * table = conn->OpenTable( "ULCustomer" );
    if( table == UL_NULL ) {
        return false;
    }
    ULTableSchema * schema = table->GetTableSchema();
    if( schema == UL_NULL ) {
        table->Close();
        return false;
    }
    ul_column_num id_cid =
        schema->GetColumnID( "cust_id" );
    ul_column_num cname_cid =
        schema->GetColumnID( "cust_name" );
    schema->Close();
    _tprintf( "\n\nTable 'ULCustomer' row contents:\n" );
}

```

```

while( table->Next() ) {
    ul_char cname[ MAX_NAME_LEN ];
    table->GetString( cname_cid, cname, MAX_NAME_LEN );
    _tprintf( "id=%d, name=%s \n", (int)table->GetInt(id_cid), cname );
}
table->Close();
return true;
}
static bool do_sync( ULConnection * conn )
{
    ul_sync_info info;
    ul_stream_error * se = &info.stream_error;

    ULDatabaseManager::EnableTcpipSynchronization();
    conn->InitSyncInfo( &info );
    info.stream = "TCPIP";
    info.version = "custdb 12.0";
    info.user_name = "50";
    info.download_only = true;
    if( !conn->Synchronize( &info ) ) {
        _tprintf( "Synchronization error \n" );
        _tprintf( "    stream_error_code is '%lu'\n", se->stream_error_code );
        _tprintf( "    system_error_code is '%ld'\n", se->system_error_code );
        _tprintf( "    error_string is ' ' );
        _tprintf( "%s", se->error_string );
        _tprintf( "\n" );
        return false;
    }
    return true;
}
int main()
{
    ULConnection * conn;
    ULDatabaseManager::Init();
    ULDatabaseManager::SetErrorCallback( MyErrorCallBack, NULL );
    conn = open_conn();
    if( conn == UL_NULL ){
        ULDatabaseManager::Fini();
        return 1;
    }

    // Main processing code goes here ...
    do_insert( conn );
    do_select( conn );
    do_sync( conn );
    conn->Close();
    ULDatabaseManager::Fini();
    return 0;
}

```

**Parent topic:** [Tutorial: Building a Windows Application using the C++ API \[page 717\]](#)

**Previous task:** [Lesson 4: Adding Synchronization to Your Application \[page 725\]](#)

## 3.4 API Reference

Use the UltraLite C++ API to develop mobile applications.

**In this section:**



[UltraLite C++ Common API Reference \[page 729\]](#)

Use functions and macros with the Embedded SQL or C++ interface.

[UltraLite C++ API Reference \[page 731\]](#)

UltraLite C++ offers a variety of API objects.

[UltraLite Embedded SQL API Reference \[page 731\]](#)

Embedded SQL applications support several UltraLite functions.

## 3.4.1 UltraLite C++ Common API Reference

Use functions and macros with the Embedded SQL or C++ interface.

### Header File

- `ulgglobal.h`

The UltraLite C++ common API reference is available in the *UltraLite - C++ Common API Reference* at <https://help.sap.com/viewer/ad706432629c4e249f93804b239a8377/LATEST/en-US>.

#### In this section:

[Macros and Compiler Directives for UltraLite C++ Applications \[page 729\]](#)

Embedded SQL and C++ applications support a variety of directives. These directives apply to both APIs unless otherwise specified.

### 3.4.1.1 Macros and Compiler Directives for UltraLite C++ Applications

Embedded SQL and C++ applications support a variety of directives. These directives apply to both APIs unless otherwise specified.

You can supply compiler directives:

- On your compiler command line. You commonly set a directive with the `/D` option. For example, to compile an UltraLite application with user authentication, a makefile for the Microsoft Visual C++ compiler may look as follows:

```
CompileOptions=/c /DPRWIN32 /Od /Zi /DWIN32 /DUL_USE_DLL
IncludeFolders= \
/I"$(VCDIR)\include" \
/I"$(SQLANY17)\SDK\Include"
sample.obj: sample.cpp
cl $(CompileOptions) $(IncludeFolders) sample.cpp
```

`VCDIR` and `SQLANY17` are environment variables that point to their respective installation directories.

- In the compiler settings window of your user interface.
- In source code. You supply directives with the `#define` statement.

**In this section:**

[UL\\_USE\\_DLL Macro \[page 730\]](#)

Sets the application to use the runtime library DLL, rather than a static runtime library.

[UNDER\\_CE Macro \[page 730\]](#)

By default, this macro is defined in all new Microsoft Visual C++ Smart Device projects.

### 3.4.1.1.1 UL\_USE\_DLL Macro

Sets the application to use the runtime library DLL, rather than a static runtime library.

#### Remarks

Applies to Microsoft Windows Mobile and Microsoft Windows applications.

### 3.4.1.1.2 UNDER\_CE Macro

By default, this macro is defined in all new Microsoft Visual C++ Smart Device projects.

#### Remarks

Applies to Microsoft Windows Mobile applications.

#### Example

```
/D UNDER_CE
```

#### Related Information

[UltraLite Application Development for Microsoft Windows Mobile \[page 708\]](#)

## 3.4.2 UltraLite C++ API Reference

UltraLite C++ offers a variety of API objects.

The following list describes some of the commonly used API objects:

### **ULDatabaseManager**

Provides methods for managing databases and connections.

### **ULConnection**

Represents a connection to an UltraLite database. You can create one or more ULConnection objects.

### **ULTable**

Provides direct access to tables in the database.

### **ULPreparedStatement, ULResultSet, and ULResultSetSchema**

Create Dynamic SQL statements, make queries, execute INSERT, UPDATE, and DELETE statements, and attain programmatic control over database result sets.

## Header File

- `ulcpp.h`

The UltraLite C++ API reference is available in the *UltraLite - C++ API Reference* at <https://help.sap.com/viewer/f22db01a54914581a1acf1b8fd359a6f/LATEST/en-US>.

## 3.4.3 UltraLite Embedded SQL API Reference

Embedded SQL applications support several UltraLite functions.

Use the EXEC SQL INCLUDE SQLCA command to include prototypes for the functions in this chapter.

## Header Files

- `mfiletransfer.h`
- `ulprotos.h`

The UltraLite Embedded SQL API reference is available in the *UltraLite - Embedded SQL API Reference* at <https://help.sap.com/viewer/328faa6a0a4f4931b7e14006b07d7a2b/LATEST/en-US>.

### In this section:

[db\\_fini Method \[page 732\]](#)

Frees resources used by the UltraLite runtime library.

[db\\_init Method \[page 733\]](#)

Initializes the UltraLite runtime library.

## Related Information

[UltraLite C++ Application Development Using Embedded SQL \[page 676\]](#)

### 3.4.3.1 db\_fini Method

Frees resources used by the UltraLite runtime library.

≡ Syntax

```
unsigned short db_fini( SQLCA * sqlca );
```

## Returns

- 0 if an error occurs during processing. The error code is set in SQLCA.
- Non-zero if there are no errors.

## Remarks

You must not make any other UltraLite library call or execute any Embedded SQL command after `db_fini` is called.

Call `db_fini` once for each SQLCA being used.

## Related Information

[db\\_init Method \[page 733\]](#)

## 3.4.3.2 db\_init Method

Initializes the UltraLite runtime library.

### ≡ Syntax

```
unsigned short db_init( SQLCA * sqlca );
```

## Returns

- 0 if an error occurs during processing (for example, during initialization of the persistent store). The error code is set in SQLCA.
- Non-zero if there are no errors. You can begin using Embedded SQL commands and functions.

## Remarks

You must call this function before you make any other UltraLite library call, and before you execute any Embedded SQL command.

Usually you should only call this function once, passing the address of the global `sqlca` variable (as defined in the `sqlca.h` header file). If you have multiple execution paths in your application, you can use more than one `db_init` call, as long as each one has a separate `sqlca` pointer. This separate SQLCA pointer can be a user-defined one, or could be a global SQLCA that has been freed using `db_fini`.

In multithreaded applications, each thread must call `db_init` to obtain a separate SQLCA. Carry out subsequent connections and transactions that use this SQLCA on a single thread.

Initializing the SQLCA also resets any settings from previously called ULEnable functions. If you re-initialize a SQLCA, you must issue any ULEnable functions the application requires.

## Related Information

[db\\_fini Method \[page 732\]](#)

# 4 UltraLite - Java Programming

This book describes the UltraLiteJ programming interface. With UltraLiteJ, you can develop and deploy database applications to Android devices.

## In this section:

[System Requirements and Supported Platforms \[page 734\]](#)

UltraLiteJ supports the Android platform.

[UltraLiteJ Application Development \[page 734\]](#)

The UltraLiteJ API provides database functionality and synchronization to your Java applications.

[Tutorial: Building an Android Application \[page 753\]](#)

This tutorial guides you through the development of an application using the UltraLiteJ API and the Eclipse environment. In this tutorial, you run the application on a Windows simulator.

[UltraLiteJ API Reference \[page 761\]](#)

UltraLiteJ has a variety of API objects.

## 4.1 System Requirements and Supported Platforms

UltraLiteJ supports the Android platform.

To develop UltraLiteJ applications, you must have a Java IDE, such as Android Studio.

### Related Information

[Supported Platforms](#)

## 4.2 UltraLiteJ Application Development

The UltraLiteJ API provides database functionality and synchronization to your Java applications.

The API contains all the methods required to connect to an UltraLite database, perform schema operations, and maintain data using SQL statements. Advanced operations, such as data encryption and synchronization, are also supported.

## i Note

The UltraLiteJ API shares a common C++ code base with UltraLite for other platforms and its behavior is similar to that of other platforms. There are a few features available for accessing tables and rows without using SQL statements for which no API is provided on Android.

### In this section:

#### [Quick Start Guide to UltraLiteJ Application Development \[page 736\]](#)

When creating an UltraLiteJ application, you typically complete several data management tasks in your application code.

#### [Android Setup Considerations \[page 736\]](#)

There are UltraLiteJ API considerations to make before developing applications for Android devices.

#### [UltraLite Database Creation and Connection Approaches \[page 737\]](#)

Java applications must connect to a database before operations can be performed on the data. An UltraLite database can be created and connected to with a specified password using the UltraLiteJ API.

#### [Quick Start Guide to Schema Operations and Data Management \[page 739\]](#)

Create, update, or retrieve tables, indexes, foreign keys, publications, and rows in your database using SQL statements and queries.

#### [Schema Information in UltraLiteJ \[page 748\]](#)

You can programmatically retrieve database schema descriptions. These descriptions are known as **schema information** and are accessible using system tables and the UltraLiteJ API schema interfaces.

#### [Error Handling in UltraLiteJ \[page 749\]](#)

You can use the `ULjException` and `SQLCode` classes to handle errors. Most UltraLite methods throw `ULjException` errors.

#### [MobiLink Data Synchronization Using UltraLiteJ \[page 750\]](#)

Data synchronization can be performed using HTTP or HTTPS network protocols. HTTPS synchronization provides secure encryption to the MobiLink server.

#### [Deploying an UltraLiteJ application for Android \[page 751\]](#)

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, method calls, and deployment files to ensure that your UltraLiteJ application runs successfully on Android smartphones.

#### [Code Examples \[page 753\]](#)

All coding examples that use the UltraLiteJ API can be found in the `%SQLANYSAMPI7%\UltraLiteJ\` directory.

## Related Information

[UltraLite Overview \[page 8\]](#)

[Benefits of UltraLite APIs for Microsoft Windows Mobile \[page 21\]](#)

## 4.2.1 Quick Start Guide to UltraLiteJ Application Development

When creating an UltraLiteJ application, you typically complete several data management tasks in your application code.

1. Import the UltraLiteJ API package into your Java file(s).  
The UltraLiteJ package name and location depends on the device you are developing applications for.
2. Create a new Configuration object to create or connect to a database.  
Configuration objects define where the client database is located or where it should be created. They also specify the username and password required to connect to the database. Variations of a Configuration object are available for different devices and for non-persistent database stores.
3. Create a new Connection object.  
Connection objects connect to a client database using the specifications defined in the Configuration object.
4. Create or modify the database schema using SQL statements, and use the PreparedStatement interface to query the database.  
You can use SQL statements to create or update tables, indexes, foreign keys, and publications for your database.  
PreparedStatement objects query the database associated with the Connection object. They accept supported SQL statements, which are passed as strings. You can use PreparedStatement objects to update the contents of the database.
5. Generate ResultSet objects.  
ResultSet objects are created when the Connection object executes a PreparedStatement containing a SQL SELECT statement. You can use ResultSet objects to obtain rows of query results to view the table contents of the database.

### Related Information

[UltraLite SQL Statements \[page 516\]](#)

## 4.2.2 Android Setup Considerations

There are UltraLiteJ API considerations to make before developing applications for Android devices.

### JAR Resource Files

When setting up an application for the UltraLiteJ API, make sure that your project is correctly configured to use the appropriate `UltraLiteJ17.jar` or `UltraLiteJNI17.jar` file.

The UltraLiteJ API is stored in the `UltraLite\UltraLiteJ\Android\UltraLiteJNI17.jar` file of your SQL Anywhere installation. You must configure your Android development project to include the `UltraLiteJNI17.jar` file in the classpath.



Use the following statement to import the UltraLiteJ package into your Java file:

```
import com.sap.ultralitejni17.*;
```

All coding samples and tutorials contained in this document assume that the above statement is specified and that you are familiar with developing Java applications in Eclipse.

## 4.2.3 UltraLite Database Creation and Connection Approaches

Java applications must connect to a database before operations can be performed on the data. An UltraLite database can be created and connected to with a specified password using the UltraLiteJ API.

### i Note

To create an UltraLite database without using the UltraLiteJ API, use SQL Central or UltraLite command line utilities.

A Configuration object is used to configure a database store. Several implementations of a Configuration object are provided. A unique implementation exists for every type of database store supported by the UltraLiteJ API. Each implementation provides a set of methods used to configure the database store.

The following table lists the available Configuration object implementations for the supported database stores:

Store type	UltraLiteJ API support
Android file system	ConfigFileAndroid interface
Non-persistent (in memory)	ConfigNonPersistent interface

After creating and configuring a Configuration object, you use a Connection object to create or connect to the database. Connection objects can also be used to perform the following operations:

### Transactions

A transaction is the set of operations between commits or rollbacks. For persistent database stores, a commit makes permanent any changes since the last commit or rollback. A rollback returns the database to the state it was in when the last commit was invoked.

Each transaction and row-level operation in UltraLite is atomic. An insert involving multiple columns either inserts data to all the columns or to none of the columns.

Transactions must be committed to the database using the commit method of the Connection object.

### Prepared SQL statements

Methods are provided by the PreparedStatement interface to handle SQL statements. A PreparedStatement can be created using the prepareStatement method of the Connection object.

### Synchronizations

A set of objects governing MobiLink synchronization is accessed from the Connection object.

### In this section:

[Creating or Connecting to a Database \[page 738\]](#)

Use the UltraLiteJ API with your Java application to create or connect to a database.

## Related Information

[UltraLite Database Creation Approaches \[page 26\]](#)

[MobiLink File Transfers \[page 88\]](#)

[UltraLite Database Unload Utility \(ulunload\) \[page 243\]](#)

### 4.2.3.1 Creating or Connecting to a Database

Use the UltraLiteJ API with your Java application to create or connect to a database.

#### Prerequisites

An existing Java application for an Android device that implements the UltraLiteJ API.

#### Procedure

1. Create a new Configuration object that references the database name and is appropriate for your platform.

In the following examples, *config* is the Configuration object name and *DBname* is the database name.

```
ConfigFileAndroid config =
    DatabaseManager.createConfigurationFileAndroid(
        "DBname.udb",
        getApplicationContext()
    );
```

2. Set database properties using methods of the Configuration object.

For example, you may set a new database password using the setPassword method:

```
config.setPassword("my_password");
```

Use the setCreationString and setConnectionString methods to set additional creation and connection parameters, respectively.

3. Create a Connection object to create or connect to the database:

```
Connection conn = DatabaseManager.createDatabase(config);
```

The DatabaseManager.createDatabase method creates the database and returns a connection to it.

In the above example, the following code is used to connect to an existing database:

```
Connection conn = DatabaseManager.connect(config);
```

The connect method finalizes the database connection process. If the database does not exist, an error is thrown.

## Results

You can execute SQL statements from your Java application to create the tables and indexes in your database but you cannot change certain database creation parameters, such as the database name, password, or page size.

## Related Information

[UltraLite Options \[page 143\]](#)

[UltraLite Connection Parameters \[page 181\]](#)

## 4.2.4 Quick Start Guide to Schema Operations and Data Management

Create, update, or retrieve tables, indexes, foreign keys, publications, and rows in your database using SQL statements and queries.

When performing schema operations to manage data, you typically perform the following tasks in your application code:

1. Perform schema operations.  
Manage and modify the schema by using SQL statements such as CREATE TABLE or CREATE INDEX on the database connection.
2. Manage row operations.  
Manage data in tables using SQL statements such as INSERT, UPDATE, or DELETE on the database connection.
3. Retrieve row data in a result set.  
Retrieve a result set using the SELECT statement, and then traverse the row data using result set navigation methods, such as *previous* and *next*.

### In this section:

[Example: Managing Database Operations on an Android Device \[page 740\]](#)

This example illustrates how to create a sample class in an Android application that uses the UltraLiteJ API to create a database and perform basic data operations.

[Schema Operations \[page 743\]](#)

General database operations, such as table creation, can be conducted using the `PreparedStatement` object.

#### [Row Operation Management \[page 744\]](#)

Row operations can be managed using the `Connection` and `PreparedStatement` objects.

#### [Row Data Retrieval \[page 747\]](#)

Data can be retrieved using the `executeQuery` method and the navigational methods in a `ResultSet` object.

### 4.2.4.1 Example: Managing Database Operations on an Android Device

This example illustrates how to create a sample class in an Android application that uses the UltraLiteJ API to create a database and perform basic data operations.

The example illustrates the following operations:

- Create a table in a database
- Insert new rows into the table
- Update a row in the table
- Delete a row from the table
- Commit changes to the database
- Select all rows from the table by creating a result set
- Traverse the result set to view the rows in the database

In addition to performing these operations, the class contains a method named `PrintText` that is used to output successful operations to the log (see the [LogCat](#) tab in Eclipse), and a `HandleError` method that is used to report errors that may occur while performing UltraLiteJ API operations.

```
package com.sampleapp;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import com.sap.ultralitejni17.*;
public class NewUltraLiteJAppActivity extends Activity {
    Connection _conn = null;
    ResultSet _departments = null;
    PreparedStatement _inserter = null;
    PreparedStatement _updater = null;
    PreparedStatement _deleter = null;
    PreparedStatement _preparer = null;
    public void PrintText(String strText) {
        Log.i("NewUltraLiteJAppActivity", strText);
    }
    public void HandleError(ULjException err) {
        Log.w("NewUltraLiteJAppActivity", "Exception: " + err.toString());
    }
    public Connection GetDatabase(String strFilename) {
        ConfigFileAndroid config = null;
        Connection dbConnection = null;
        try {
            config = DatabaseManager.createConfigurationFileAndroid(
                strFilename, getApplicationContext()
            );
            dbConnection = DatabaseManager.connect(config);
        }
    }
}
```

```

        PrintText("Successfully connected to the database at: "
            + strFilename);
    } catch(ULjException ex) {
        if (config != null) {
            try {
                dbConnection = DatabaseManager.createDatabase(config);
                PrintText("Successfully created a new database at: "
                    + strFilename);
            } catch(ULjException exception) {
                HandleError(exception);
            }
        }
        HandleError(ex);
    }
    return dbConnection;
}
public void Commit() {
    try {
        _conn.commit();
    } catch (ULjException e1) {
        HandleError(e1);
    }
}
public void CloseDatabase() {
    try {
        _conn.release();
    } catch (ULjException e) {
        HandleError(e);
    }
}
public void ExecutesSQLStatement(String strSQLstmt) {
    PreparedStatement ps;
    try {
        ps = _conn.prepareStatement(strSQLstmt);
        ps.execute();
        ps.close();
        PrintText("Successfully executed: " + strSQLstmt);
    } catch (ULjException e) {
        HandleError(e);
    }
}
public void InitStatements() {
    String stmt;
    try {
        stmt = "INSERT INTO Department(dept_no, name) VALUES (?,?)";
        _inserter = _conn.prepareStatement(stmt);
        stmt = "UPDATE Department SET dept_no = ? WHERE dept_no = ?";
        _updater = _conn.prepareStatement(stmt);
        stmt = "DELETE FROM Department WHERE dept_no = ?";
        _deleter = _conn.prepareStatement(stmt);
    } catch (ULjException e) {
        HandleError(e);
    }
}
public void FiniStatements() {
    try {
        _departments.close();
        _inserter.close();
        _updater.close();
        _deleter.close();
        _preparer.close();
    } catch (ULjException e) {
        HandleError(e);
    }
}
public void AddDepartment(int deptID, String deptName) {
    try {
        _inserter.set(1, deptID);

```

```

        _inserter.set(2, deptName);
        _inserter.execute();
        PrintText("Successfully executed:"
            + " INSERT INTO Department(dept_no, name)"
            + " VALUES (" + deptID + ", " + deptName + ")");
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void UpdateDepartment(int deptIDold, int deptIDnew) {
    try {
        _updater.set(1, deptIDnew);
        _updater.set(2, deptIDold);
        _updater.execute();
        PrintText("Successfully executed:"
            + " UPDATE Department SET dept_no = " + deptIDnew
            + " WHERE dept_no = " + deptIDold);
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void DeleteDepartment(int deptID) {
    try {
        _deleter.set(1, deptID);
        _deleter.execute();
        PrintText("Successfully executed:"
            + " DELETE FROM Department WHERE dept_no = " + deptID);
    } catch (ULjException e) {
        HandleError(e);
    }
}

public ResultSet SelectDepartmentRows() {
    String stmt = "SELECT * FROM Department ORDER BY dept_no";
    _preparer = null;
    _departments = null;
    try {
        _preparer = _conn.prepareStatement(stmt);
        _departments = _preparer.executeQuery();
        PrintText("Successfully executed: " + stmt);
    } catch (ULjException e) {
        HandleError(e);
    }
    return _departments;
}

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    PrintText("Starting application...");

    _conn = GetDatabase("test.udb");
    if (_conn == null) {
        return;
    }

    String[] stmt = new String[3];

    stmt[0] = "CREATE TABLE Department("
        + "dept_no INT PRIMARY KEY, "
        + "name CHAR(50) NOT NULL)";
    stmt[1] = "CREATE TABLE Employee("
        + "id INT PRIMARY KEY, "
        + "last_name CHAR(50) NOT NULL, "
        + "first_name CHAR(50) NOT NULL, "
        + "dept_id INT NOT NULL, "
        + "NOT NULL FOREIGN KEY(dept_id) "
        + "REFERENCES Department(dept_no))";
    stmt[2] = "CREATE INDEX ON Employee(last_name, first_name)";
}

```

```

for(int i = 0; i< stmt.length; i++) {
    ExecuteSQLStatement(stmt[i]);
}
InitStatements();
AddDepartment(101, "Electronics");
AddDepartment(105, "Sales");
AddDepartment(109, "Accounting");
UpdateDepartment(101, 102);
DeleteDepartment(102);
Commit();
_departments = SelectDepartmentRows();
if (_departments != null) {
    try {
        while(_departments.next()) {
            int dept_no = _departments.getInt(1);
            String dept_name = _departments.getString(2);
            PrintText("Department no.:" + dept_no
                + " Department name: " + dept_name);
        }
    } catch (ULjException e) {
        HandleError(e);
    }
}

FiniStatements();
CloseDatabase();
PrintText("Closing application...");
finish();
}
}

```

## Related Information

[UltraLite SQL Statements \[page 516\]](#)

### 4.2.4.2 Schema Operations

General database operations, such as table creation, can be conducted using the PreparedStatement object.

Perform schema operations by following these general tasks:

1. Construct a SQL statement in a String variable.
2. Create a PreparedStatement object by passing the String variable to the Connection.prepareStatement method.
3. Call the PreparedStatement.execute method to perform the operation on the database.
4. Close the PreparedStatement object to free resources.

### Example

The code referenced in this example is part of a complete sample that illustrates how to perform basic schema and data management operations using the UltraLiteJ API.

Individual CREATE TABLE and CREATE INDEX statements are constructed and passed to a custom method named ExecuteSQLStatement to perform all necessary schema operations:

```
String[] stmt = new String[3];
stmt[0] = "CREATE TABLE Department("
    + "dept_no INT PRIMARY KEY, "
    + "name CHAR(50) NOT NULL)";
stmt[1] = "CREATE TABLE Employee("
    + "id INT PRIMARY KEY, "
    + "last_name CHAR(50) NOT NULL, "
    + "first_name CHAR(50) NOT NULL, "
    + "dept_id INT NOT NULL, "
    + "NOT NULL FOREIGN KEY(dept_id) "
    + "REFERENCES Department(dept_no))";
stmt[2] = "CREATE INDEX ON Employee(last_name, first_name)";

for(int i = 0; i < stmt.length; i++) {
    ExecuteSQLStatement(stmt[i]);
}
```

The ExecuteSQLStatement method consists of the following code:

```
public void ExecuteSQLStatement(String strSQLstmt) {
    PreparedStatement ps;
    try {
        ps = _conn.prepareStatement(strSQLstmt);
        ps.execute();
        ps.close();
        PrintText("Successfully executed: " + strSQLstmt);
    } catch (ULjException e) {
        HandleError(e);
    }
}
```

## Related Information

[Example: Managing Database Operations on an Android Device \[page 740\]](#)

### 4.2.4.3 Row Operation Management

Row operations can be managed using the Connection and PreparedStatement objects.

Manage row operations by performing the following general tasks:

1. Construct a SQL statement in a String variable.
2. Create a PreparedStatement object by passing the String variable to the Connection.prepareStatement method.
3. Set any host variables, indicated by the question mark (?) character, by using the PreparedStatement.set method.  
Each host variable can be referenced in accordance to its ordinal position in the statement. For example, the first ? is referenced as 1, and the second as 2. The [set](#) method, illustrated in the example below, allows you to reference the ordinal position of the variable and specify a new value.
4. Call the PreparedStatement.execute method to perform the operation on the database.



- Commit the changes to the database by calling the `Connection.commit` method to make the changes permanent; otherwise, call the `Connection.rollback` method.  
Transactions must be explicitly committed or rolled back by using the methods supported by the `Connection` interface.
- Close the `PreparedStatement` object to free resources.

## Example

The code referenced in this example is part of a complete sample that illustrates how to perform basic schema and data management operations using the UltraLiteJ API.

Global `PreparedStatement` objects are defined and instantiated by an `InitStatements` method call. Data insert, update, and delete operations are illustrated in the custom `AddDepartment`, `UpdateDepartment`, and `DeleteDepartment` methods, respectively. The `Commit` method illustrates how to make the row operations permanent. The `FinisStatements` method closes the global `PreparedStatement` objects and frees resources.

```

Connection _conn = null;
ResultSet _departments = null;
PreparedStatement _inserter = null;
PreparedStatement _updater = null;
PreparedStatement _deleter = null;
PreparedStatement _preparer = null;
public void InitStatements() {
    String stmt;
    try {
        stmt = "INSERT INTO Department(dept_no, name) VALUES (?,?)";
        _inserter = _conn.prepareStatement(stmt);
        stmt = "UPDATE Department SET dept_no = ? WHERE dept_no = ?";
        _updater = _conn.prepareStatement(stmt);
        stmt = "DELETE FROM Department WHERE dept_no = ?";
        _deleter = _conn.prepareStatement(stmt);
    } catch (ULjException e) {
        HandleError(e);
    }
}
public void FinisStatements() {
    try {
        _departments.close();
        _inserter.close();
        _updater.close();
        _deleter.close();
        _preparer.close();
    } catch (ULjException e) {
        HandleError(e);
    }
}
public void AddDepartment(int deptID, String deptName) {
    try {
        _inserter.set(1, deptID);
        _inserter.set(2, deptName);
        _inserter.execute();
        PrintText("Successfully executed:"
            + " INSERT INTO Department(dept_no, name)"
            + " VALUES (" + deptID + ", " + deptName + ")");
    } catch (ULjException e) {
        HandleError(e);
    }
}
public void UpdateDepartment(int deptIDold, int deptIDnew) {
    try {

```

```

        _updater.set(1, deptIDnew);
        _updater.set(2, deptIDold);
        _updater.execute();
        PrintText("Successfully executed:"
            + " UPDATE Department SET dept_no = " + deptIDnew
            + " WHERE dept_no = " + deptIDold);
    } catch (ULjException e) {
        HandleError(e);
    }
}
public void DeleteDepartment(int deptID) {
    try {
        _deleter.set(1, deptID);
        _deleter.execute();
        PrintText("Successfully executed:"
            + " DELETE FROM Department WHERE dept_no = " + deptID);
    } catch (ULjException e) {
        HandleError(e);
    }
}
public void Commit() {
    try {
        _conn.commit();
    } catch (ULjException e1) {
        HandleError(e1);
    }
}
}

```

### i Note

String concatenation is recommended over host variable usage when constructing SQL statements that must be executed only once.

For example, the following code, which uses String concatenation to construct SQL statements, can be used to replace the DeleteDepartment method:

```

public void DeleteDepartment(int deptID) {
    String stmt = "DELETE FROM Department WHERE dept_no = " + deptID;
    PreparedStatement deleter;
    try {
        deleter = _conn.prepareStatement(stmt);
        deleter.execute();
        deleter.close();
        PrintText("Successfully executed: " + stmt);
    } catch (ULjException e) {
        HandleError(e);
    }
}
}

```

## Related Information

[Example: Managing Database Operations on an Android Device \[page 740\]](#)

## 4.2.4.4 Row Data Retrieval

Data can be retrieved using the `executeQuery` method and the navigational methods in a `ResultSet` object.

Retrieve row data from a table by performing the following general tasks:

1. Construct a SELECT SQL statement in a String variable.
2. Create a `PreparedStatement` object by passing the String variable to the `Connection.prepareStatement` method.
3. Call the `PreparedStatement.executeQuery` method to assign the query results to a `ResultSet` object.
4. Traverse through the `ResultSet` object using the navigational methods to retrieve the row data.

The following navigational methods can be used to traverse a result set:

### **afterLast**

Position immediately after the last row.

### **beforeFirst**

Position immediately before the first row.

### **first**

Move to the first row.

### **last**

Move to the last row.

### **next**

Move to the next row.

### **previous**

Move to the previous row.

### **relative(offset)**

Move a specified number of rows relative to the current row, as specified by the signed offset value. Positive offset values move forward in the result set, relative to the current pointer position in the result set. Negative offset values move backward in the result set. An offset value of zero does not move the current location, but allows you to repopulate the row buffer.

5. Close the `ResultSet` and `PreparedStatement` objects to free resources.

## Example

The code referenced in this example is part of a complete sample that illustrates how to perform basic schema and data management operations using the UltraLiteJ API.

A result set is retrieved in the custom `SelectDepartmentRows` method:

```
public ResultSet SelectDepartmentRows() {
    String stmt = "SELECT * FROM Department ORDER BY dept_no";
    _preparer = null;
    _departments = null;
    try {
        _preparer = _conn.prepareStatement(stmt);
        _departments = _preparer.executeQuery();
        PrintText("Successfully executed: " + stmt);
    }
}
```

```

    } catch (ULjException e) {
        HandleError(e);
    }
    return _departments;
}

```

Row data is retrieved by traversing through a result set using the *next* navigational method:

```

_departmentments = SelectDepartmentRows();
if (_departmentments != null) {
    try {
        while(_departmentments.next()) {
            int dept_no = _departmentments.getInt(1);
            String dept_name = _departmentments.getString(2);
            PrintText("Department no.:" + dept_no
                + " Department name: " + dept_name);
        }
    } catch (ULjException e) {
        HandleError(e);
    }
}
}

```

## Related Information

[Example: Managing Database Operations on an Android Device \[page 740\]](#)

## 4.2.5 Schema Information in UltraLiteJ

You can programmatically retrieve database schema descriptions. These descriptions are known as **schema information** and are accessible using system tables and the UltraLiteJ API schema interfaces.

### Accessing Schema Information Using System Tables

Database schema information is stored in UltraLite system tables. You can access this information by executing a regular SQL query to select the desired information from the appropriate system table, and then accessing the result set.

### Accessing Schema Information Using Schema Interfaces

Some schema information can be accessed using schema interfaces instead of system tables. The UltraLiteJ API contains the following schema interfaces:

#### **TableSchema**

Returns information about the column and index configurations.

#### **IndexSchema**

Returns information about the columns in the index. IndexSchema objects can be retrieved from TableSchema objects.

### ColumnSchema

Returns information about the columns in the table. ColumnSchema objects can be retrieved from TableSchema objects.

## Related Information

[Row Data Retrieval \[page 747\]](#)

[UltraLite System Tables \[page 248\]](#)

## 4.2.6 Error Handling in UltraLiteJ

You can use the ULjException and SQLCode classes to handle errors. Most UltraLite methods throw ULjException errors.

SQLCode errors are negative numbers indicating the error type, and can be referenced using constants such as ULjException.SQLE\_INDEX\_NOT\_FOUND. You can use the ULjException.getErrorCode method to retrieve the SQLCode value assigned to the error. You can use the ULjException.toString method to obtain a descriptive text of the error.

### Example

The following example illustrates a Java class that uses the ULjException class to handle an error that may occur when connecting to an UltraLite database:

```
import com.sap.ultralitejni17.*;
import java.util.log;
import android.content.Context;
class DataAccess {
    DataAccess() {
    }

    public static synchronized DataAccess getDataAccess(boolean reset)
        throws Exception
    {
        if (_da == null) {
            _da = new DataAccess();
            ConfigFileAndroid config =
DatabaseManager.createConfigurationFileAndroid("HelloDB.udb",
getApplicationContext());
            if (reset) {
                _conn = DatabaseManager.createDatabase(config);
            }
            else {
                try {
                    _conn = DatabaseManager.connect(config);
                }
                catch (ULjException uex1) {
```

```

        if (uex1.getErrorCode() !=
        ULjException.SQLE_ULTRALITE_DATABASE_NOT_FOUND) {
            Log.e("Exception: " + uex1.toString(), ". Recreating
        database...");
        }
        _conn = DatabaseManager.createDatabase(config);
    }
}
return _da;
}
private static Connection _conn;
private static DataAccess _da;
}

```

## Related Information

[SQL Anywhere Error Messages Sorted by SQLCODE](#)

## 4.2.7 MobiLink Data Synchronization Using UltraLiteJ

Data synchronization can be performed using HTTP or HTTPS network protocols. HTTPS synchronization provides secure encryption to the MobiLink server.

To synchronize data, your application must perform the following steps:

1. Instantiate a syncParms object, which contains information about the consolidated database (name of the server, port number), the name of the database to be synchronized, and the definition of the tables to be synchronized.
2. Call the synchronize method from the connection object with the syncParms object to perform the synchronization.

The data to be synchronized can be defined at the table level. You cannot configure synchronization for portions of a table.

The CustDB tutorial demonstrates data synchronization with an UltraLiteJ application and can be found in [%SQLANYSAMP17%\UltraLiteJ\Android\CustDB](#).

### In this section:

[Network Protocol Options for UltraLiteJ Synchronization Streams \[page 751\]](#)

When synchronizing with a MobiLink server, you must set the network protocol in your application. Each database synchronizes over a network protocol. Two network protocols are available for UltraLiteJ: HTTP and HTTPS.

## Related Information

[Tutorial: Building an Android Application \[page 753\]](#)

## 4.2.7.1 Network Protocol Options for UltraLiteJ Synchronization Streams

When synchronizing with a MobiLink server, you must set the network protocol in your application. Each database synchronizes over a network protocol. Two network protocols are available for UltraLiteJ: HTTP and HTTPS.

For the network protocol you set, choose from a set of corresponding protocol options to ensure that the UltraLiteJ application can locate and communicate with the MobiLink server. Network protocol options provide information such as addressing information (host and port) and protocol-specific information.

### Setting up an HTTP Network Protocol

An HTTP network protocol is set with the `StreamHTTPParms` interface in the UltraLiteJ API. Use the interface methods to specify the network protocol options defined on the MobiLink server.

### Setting up an HTTPS Network Protocol

An HTTPS network protocol is set with the `StreamHTTPSParms` interface in the UltraLiteJ API. Use the interface methods to specify the network protocol options defined on the MobiLink server.

### Related Information

[MobiLink Client Network Protocol Options](#)

## 4.2.8 Deploying an UltraLiteJ application for Android

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, method calls, and deployment files to ensure that your UltraLiteJ application runs successfully on Android smartphones.

### Procedure

1. Add the following files to your Android project:

Copy From	Copy To
%SQLANY17%\UltraLite\UltraLiteJ\Android \UltraLiteJNI17.jar	app\libs
%SQLANY17%\UltraLite\UltraLiteJ\Android \x86\libultralitej17.so	app\src\main\jniLibs\arm64-v8a
%SQLANY17%\UltraLite\UltraLiteJ\Android \x86_64\libultralitej17.so	app\src\main\jniLibs\x86
%SQLANY17%\UltraLite\UltraLiteJ\Android \ARM\libultralitej17.so	app\src\main\jniLibs\armeabi-v7a
%SQLANY17%\UltraLite\UltraLiteJ\Android \ARM64\libultralitej17.so	app\src\main\jniLibs\arm64-v8a

- Specify the following parameters when encrypting the database:
  - When using AES encryption, set the connection parameter `DBKEY=encryption-key` while creating and connecting to the database.

To set these parameters, use the `setCreationString` and `setConnectionString` methods.

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

Synchronization type	Parameter settings
TCP/IP	Set the <code>Stream</code> synchronization parameter to "tcpip".
HTTP	Set the <code>Stream</code> synchronization parameter to "http".
RSA TLS	Set the <code>Stream</code> synchronization parameter to "tls".
RSA HTTPS	Set the <code>Stream</code> synchronization parameter to "https".

- When using RSA end-to-end encryption (RSA E2EE), set the protocol option `e2ee_public_key=key-file`.
- When using ZLIB compression, set the protocol option `compression=zlib`.
- When using AES encryption, call the `ConfigPersistent.EnableAesDBEncryption` method.

## Results

The UltraLiteJ application runs successfully on the Android device that it is deployed to.

## Next Steps

Deploy an UltraLite database to the Android mobile device that the application was deployed to, or create a new database with the deployed application.



## Related Information

[UltraLite Database Deployment Techniques \[page 130\]](#)

[UltraLite Application Build and Deployment Specifications \[page 124\]](#)

### 4.2.9 Code Examples

All coding examples that use the UltraLiteJ API can be found in the `%SQLANYSAMPI7%\UltraLiteJ\` directory.

A sample Eclipse project that uses the CustDB sample database is available in the `%SQLANYSAMPI7%\UltraLiteJ\Android\CustDB` directory. The source code can be found in `%SQLANYSAMPI7%\UltraLiteJ\Android\CustDB\src\com\sap\custdb`.

A tutorial based on the example is available.

## 4.3 Tutorial: Building an Android Application

This tutorial guides you through the development of an application using the UltraLiteJ API and the Eclipse environment. In this tutorial, you run the application on a Windows simulator.

### Prerequisites

You need the following:

- Familiarity with Java
- Familiarity with Eclipse

You require the following software:

- Eclipse 3.5.2 or later
- Android SDK Starter Package
- Android Development Tools (ADT) Plug-in for Eclipse 1.1 or later
- SQL Anywhere 17 samples
- UltraLiteJ API

### Context

The Android application used in this tutorial is located in the `%SQLANYSAMPI7%\UltraLiteJ\Android\Eclipse\CustDB` directory.

## i Note

An alternative Android Studio version of this tutorial is located in the `%SQLANYXSAMP17%\UltraLiteJ\Android\AndroidStudio\CustDB\` directory.

The application code, located in the `src\com\sap\custdb` directory, references the UltraLiteJ API to perform the following tasks:

- UltraLite remote database creation.
- SQL operations on the database.
- Data synchronization with the SQL Anywhere CustDB sample database using MobiLink.

The `res\menu` and `res\layout` directories illustrate how to create Android menu items and interfaces. You can view these files through Eclipse when you create the new Android project.

The `AndroidManifest.xml` project file was modified so that the Android application can access the network, which is required for data synchronization. The following permission statement was added:

```
<uses-permission android:name="android.permission.INTERNET" />
```

1. [Lesson 1: Setting up a New Android Project \[page 755\]](#)  
In this lesson, you create a new Android project through the Eclipse Integrated Development Environment.
2. [Lesson 2: Starting the MobiLink Server \[page 757\]](#)  
In this lesson, you start the MobiLink server to perform synchronization.
3. [Lesson 3: Running Your Android Application \[page 758\]](#)  
In this lesson, you run your application through an Android simulator.
4. [Lesson 4: Testing Your Android Application and Synchronizing \[page 759\]](#)  
In this lesson, you use your Android application to update the UltraLite remote database and synchronize the CustDB consolidated database.
5. [Lesson 5: Cleaning up \[page 761\]](#)  
Remove your recently created tutorial materials from your computer.

## Related Information

[Android SDK and the Android Development Tools \(ADT\) Plug-in for Eclipse](#)   
[UltraLiteJ API Reference \[page 761\]](#)

## 4.3.1 Lesson 1: Setting up a New Android Project

In this lesson, you create a new Android project through the Eclipse Integrated Development Environment.

### Prerequisites

This lesson assumes that you have installed the required software.

### Context

This tutorial assumes that you are familiar with Java and Eclipse.

### Procedure

1. Copy the Android libraries to your Android CustDB sample directory.

Open a command prompt, change to the `%SQLANYSAMP17%\UltraLiteJ\Android\CustDB\` directory, and then run the following command:

```
setup.bat
```

The `UltraLiteJNI17.jar` and `libultralitej17.so` files are copied into the `Android\CustDB\libs` and `Android\CustDB\libs\armeabi` directories.

2. Run Eclipse.

The default application path is `C:\Eclipse\eclipse.exe`.

3. In the *Workspace* field, specify a working directory that is not your CustDB sample directory, and then click *OK*.
4. Import the CustDB project into Eclipse.
  - a. Click **File** > *Import*.
  - b. Expand the *General* directory, and then click *Existing Projects into Workspace*. Click *Next*.
  - c. In the *Select Root Directory* field, type `%SQLANYSAMP17%\UltraLiteJ\Android\CustDB`. Select *Copy Projects Into Workspace*, and then click *Finish*.
5. Make sure that the appropriate Android SDK path is specified in Eclipse.

#### Note

You must install an Android SDK before specifying the path.

- a. Click **Window** > *Preferences*.
- b. In the left pane, click *Android*.

- c. In the *SDK Location* field, type the location of the Android SDK and then click *Apply*.  
A list of available build targets appears.
  - d. Click *OK*.
6. Make sure that the UltraLiteJ library path is specified in Eclipse.
  - a. Click **File > Properties**.
  - b. In the left pane, click **Java Build Path > User libraries**.
  - c. Click the *Libraries* tab.
  - d. Click *UltraLiteJNI17.jar*, and then click *Edit*.
  - e. From your working directory, open `\CustDB\libs\UltraLiteJNI17.jar`.
  - f. Click *OK*.
7. Add the path to your UltraLiteJNI Javadoc documentation to the project.
  - a. In the left pane, click *Javadoc Location*.
  - b. Click *Browse*, and then open `%SQLANY17%\UltraLite\UltraLiteJ\Android\html`.
  - c. Click *OK*.
  - d. Click *OK* to close the window.
8. Build the project. Click **Project > Clean**, and then click *OK*.  
The project should build without errors occurring, but you may notice warnings listed under the *Problems* tab.

## Results

The UltraLiteJ API is functional in the new Android application.

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building an Android Application \[page 753\]](#)

**Next task:** [Lesson 2: Starting the MobiLink Server \[page 757\]](#)

## 4.3.2 Lesson 2: Starting the MobiLink Server

In this lesson, you start the MobiLink server to perform synchronization.

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

### Procedure

Start MobiLink by running the following command from `%SQLANYSAMPI7%\MobiLink\CustDB\`:

```
mlsrv17 -v+ -zu+ -c "DSN=SQL Anywhere 17 CustDB;UID=ml_server;PWD=sql" -x  
http(port=80) -ot ml.mls
```

The `-c` option connects MobiLink to the SQL Anywhere CustDB database. The `-v+` option sets a high level of verbosity so that you can follow what is happening in the MobiLink server messages window. The `-x` option specifies the port number being used for the communications. The `-ot` option specifies that a log file (`ml.mls`) is to be created in the directory where you started the MobiLink server. The Android app uses HTTP to connect to MobiLink. If you use another port besides port 80, then you must open that port in your firewall.

### Results

The MobiLink server has started.

### Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building an Android Application \[page 753\]](#)

**Previous task:** [Lesson 1: Setting up a New Android Project \[page 755\]](#)

**Next task:** [Lesson 3: Running Your Android Application \[page 758\]](#)

## Related Information

[MobiLink Server Options](#)

### 4.3.3 Lesson 3: Running Your Android Application

In this lesson, you run your application through an Android simulator.

#### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

#### Procedure

1. Set up your Android virtual device in Eclipse.
  - a. Click **Window** > **AVD Manager**.
  - b. Click **New**.  
  
The *Create New Android Virtual Device (AVD)* window appears.
  - c. In the *Name* field, type **my\_avd**.
  - d. In the *Target* field, click **Android 2.2 - API Level 8**.
  - e. Click **Create AVD**.
  - f. Close the *AVD Manager* window.
2. In the *Package Explorer* window, select **CustDB**.
3. From the *Run* menu, choose **Run As** > **Android Application**.

The Android simulator loads.

4. Click **Menu**.

Your Android application loads.

#### Results

The UltraLite application loads in a simulated Android device.

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building an Android Application \[page 753\]](#)

**Previous task:** [Lesson 2: Starting the MobiLink Server \[page 757\]](#)

**Next task:** [Lesson 4: Testing Your Android Application and Synchronizing \[page 759\]](#)

## 4.3.4 Lesson 4: Testing Your Android Application and Synchronizing

In this lesson, you use your Android application to update the UltraLite remote database and synchronize the CustDB consolidated database.

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

### Procedure

1. Ensure that the *Employee ID* field is 50, the *Host* field is 10.0.2.2, and the *Port* field is 80, and then click *Save*.

The application automatically synchronizes and a set of customers, products, and orders is downloaded to the application from the CustDB consolidated database.

2. In the simulator, click **Menu** > **New**.
3. In the *Customer* field, choose *Ace Properties*.
4. In the *Product* field, choose *4x8 Drywall x100*.
5. In the *Quantity* field, type *999*.
6. In the *Discount* field, type *25*.
7. Click *OK* to add the new order.
8. Synchronize the application with the CustDB consolidated database.

In the simulator, click *Menu* and then click *Sync*.

9. Connect to the CustDB consolidated database with Interactive SQL.

- a. Click **Start** > **Programs** > **SQL Anywhere 17** > **Administration Tools** > **Interactive SQL**, or run the following command:

```
dbisql
```

- b. Click **ODBC Data Source Name**, click **Browse**, click **SQL Anywhere 17 CustDB**, and then click **OK**.
- c. In the password field, type **sql** and then click **Connect**.

10. Verify that the synchronization was successful.

Execute the following SQL statement in Interactive SQL:

```
SELECT order_id, disc, quant, notes, status, c.cust_id,  
       cust_name, p.prod_id, prod_name, price  
FROM ULOrder o, ULCustomer c, ULProduct p  
WHERE o.cust_id = c.cust_id  
AND o.prod_id = p.prod_id  
AND c.cust_name = 'Ace Properties'  
AND p.prod_name = '4x8 Drywall x100'
```

Synchronization was successful when an order entry appears in Interactive SQL.

11. Close the simulator window.

## Results

The changes you made in the simulator are synchronized with the CustDB consolidated database.

## Next Steps

Proceed to the next lesson.

**Task overview:** [Tutorial: Building an Android Application \[page 753\]](#)

**Previous task:** [Lesson 3: Running Your Android Application \[page 758\]](#)

**Next task:** [Lesson 5: Cleaning up \[page 761\]](#)



## 4.3.5 Lesson 5: Cleaning up

Remove your recently created tutorial materials from your computer.

### Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

### Procedure

1. Close Eclipse.

Click **File > Exit**.

2. Close MobiLink, Interactive SQL, and synchronization client windows by right-clicking each task bar item and clicking *Exit* or *Shut Down*.
3. Reset the CustDB database.

Run the following command from the `%SQLANYSAMPI7%\UltraLite\CustDB` directory:

```
makepbs
```

### Results

The materials are removed from your computer, and this tutorial can be repeated again from the first lesson.

**Task overview:** [Tutorial: Building an Android Application \[page 753\]](#)

**Previous task:** [Lesson 4: Testing Your Android Application and Synchronizing \[page 759\]](#)

## 4.4 UltraLiteJ API Reference

UltraLiteJ has a variety of API objects.

The following list describes some of the commonly used API objects:

#### **DatabaseManager**

Provides methods for managing databases and connections.

### **Connection**

Represents a connection to an UltraLite database. You can create one or more Connection objects.

### **SyncParms**

Synchronizes your UltraLite database with a MobiLink server.

### **PreparedStatement, ResultSet**

Create dynamic SQL statements, make queries, execute INSERT, UPDATE, and DELETE statements, and attain programmatic control over database result sets.

## **Package [Android]**

```
com.sap.ultralitejni17
```

# 5 UltraLite - UWP Programming

This book describes the UltraLite UWP (Universal Windows Platform) programming interface. UltraLite for UWP provides database functionality and synchronization to Microsoft Windows 10 and Microsoft Windows 10 Mobile UWP apps.

## In this section:

[System Requirements and Supported Platforms \[page 763\]](#)

UltraLite for UWP supports Microsoft Windows 10.

[UltraLite for UWP Application Development \[page 764\]](#)

UltraLite for UWP provides database functionality and synchronization to Microsoft Windows Phone or Windows Store Apps.

[Tutorial: Building a Windows Phone Application \[page 770\]](#)

Develop an application for Windows Phone by using the UltraLite for UWP API and the Microsoft Visual Studio environment. This tutorial assumes that the application runs on a Windows Phone emulator.

[UltraLite for UWP API Reference \[page 775\]](#)

UltraLite for UWP has a variety of API objects that you use from C#, C++, or JavaScript.

## 5.1 System Requirements and Supported Platforms

UltraLite for UWP supports Microsoft Windows 10.

### System requirements

To develop UltraLite for UWP applications, you must have the following software:

#### Windows 10

- Windows 10 SDK
- Microsoft Visual Studio 2017

### Target platforms

UltraLite for UWP supports the following target platforms:

- Windows 10 Universal Windows Platform (UWP), x86, x64, or ARM CPU

## Related Information

[Supported Platforms](#)

[Windows Phone SDK 8.0 download center](#) 

## 5.2 UltraLite for UWP Application Development

UltraLite for UWP provides database functionality and synchronization to Microsoft Windows Phone or Windows Store Apps.

The API contains all the methods required to connect to an UltraLite database, perform schema operations, maintain data using SQL statements, and encrypt data.

There is a CustDb sample UltraLite for UWP. The sample is a Visual Studio 2015 project that targets the Universal Windows Platform (UWP). The project is located in %SQLANYXSAMP17%\UltraLite.WinRT\Windows10UWP\CustDb.

### In this section:

[Quick Start Guide to UltraLite for UWP Application Development \[page 765\]](#)

When creating an application, you typically complete several data management tasks in your application code.

[UltraLite for UWP Setup Considerations \[page 765\]](#)

There are API considerations to make before developing UWP applications.

[Quick Start Guide to Schema Operations and Data Management \[page 766\]](#)

Create or retrieve tables, indexes, foreign keys, publications, and rows in your database using SQL statements and queries.

[Deploying an UltraLite Application for Windows Phone or Windows Store Apps \[page 769\]](#)

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, link libraries, method calls, and deployment files to ensure that your UltraLite application runs successfully on Windows Phone devices or Windows Store Apps.

## Related Information

[UltraLite Overview \[page 8\]](#)

[Benefits of UltraLite APIs for Microsoft Windows Mobile \[page 21\]](#)

## 5.2.1 Quick Start Guide to UltraLite for UWP Application Development

When creating an application, you typically complete several data management tasks in your application code.

UltraLite for UWP uses the Windows Phone Runtime API model. You implement your application using C++ or C# and then use the API projection into the required language.

1. Initialize a DatabaseManager object.
2. Open a connection to an existing database or create a new database using methods of the Database Manager object.
3. Finalize the DatabaseManager object when the application terminates.

### Related Information

[UltraLite for UWP Setup Considerations \[page 765\]](#)

[UltraLite SQL Statements \[page 516\]](#)

## 5.2.2 UltraLite for UWP Setup Considerations

There are API considerations to make before developing UWP applications.

### Secure Data Synchronization

Due to limitations of the Windows Phone Runtime APIs, secure data synchronization has the following caveats:

- HTTPS must be used for the network protocol
- Client certificates are not supported
- The trusted\_certificate=filename protocol option is not supported. The application must add trusted root certificates to the trusted certificate store on the device before it can perform HTTPS synchronization operations.

The following MobiLink network protocol options are not supported by the client:

- certificate\_company
- certificate\_name
- client\_port
- e2ee\_public\_key
- fips
- identity
- identity\_name
- identity\_password

- identity\_adapter\_name
- network\_leave\_open
- network\_name
- trusted\_certificates
- trusted\_certificate\_name

## Windows Phone Application Settings

The following capabilities need to be defined in the Universal Windows Platform application's package manifest file:

`<Capability name="ID_CAP_IDENTITY_DEVICE"/>` This capability generates a unique device identifier, which is used to generate good quality GUIDs.

`<Capability name="ID_CAP_NETWORKING"/>` This capability is required to perform network operations.

## Windows Store Apps Settings

The following capabilities need to be defined in the Universal Windows Platform application's package manifest file:

`<Capability name="internetClient"/>` This capability is located in package.appxmanifest.

## Related Information

[UltraLite Database Creation Approaches \[page 26\]](#)

## 5.2.3 Quick Start Guide to Schema Operations and Data Management

Create or retrieve tables, indexes, foreign keys, publications, and rows in your database using SQL statements and queries.

When performing schema operations to manage data, you typically perform the following tasks in your application code:

1. Perform schema operations.  
Manage and modify the schema by using SQL statements such as CREATE TABLE or CREATE INDEX on the database connection.
2. Manage row operations.  
Manage data in tables using the INSERT SQL statement on the database connection.

3. Retrieve row data in a result set.  
Retrieve a result set using the SELECT statement, and then traverse the row data using result set navigation methods, such as previous and next.

**In this section:**

[Example: Managing Database Operations on a Windows Phone \[page 767\]](#)

This example illustrates how to create a sample class in a Windows Phone application to create a database and perform basic data operations.

## Related Information

[Example: Managing Database Operations on a Windows Phone \[page 767\]](#)

### 5.2.3.1 Example: Managing Database Operations on a Windows Phone

This example illustrates how to create a sample class in a Windows Phone application to create a database and perform basic data operations.

The example illustrates the following operations:

- Create a table in a database
- Insert new rows into the table
- Update a row in the table
- Delete a row from the table
- Commit changes to the database
- Select all rows from the table by creating a result set
- Traverse the result set to view the rows in the database

```
using UltraLite;
using Windows.Storage;

if (DatabaseManager.Init()) {
    // DatabaseManager.Init() succeeded.
} else {
    // DatabaseManager.Init() failed.
}
// Use DatabaseManager.Fini() when terminating the app.
private async void Button1_Click_1(object sender, RoutedEventArgs e)
{
    String mypath = ApplicationData.Current.LocalFolder.Path;
    Connection conn = null;
    int code;
    String errorParms;
    try {
        conn = DatabaseManager.CreateDatabase("dbf=" + mypath +
            "\\test.udb", "");
        conn.ExecuteStatement("CREATE TABLE T
            (c1 integer primary key, c2 char(20))");
    }
```

```

        PreparedStatement ps = conn.PrepareStatement
            ("INSERT INTO T VALUES(?,?)");
        ps.SetParameterInt32(0, 11);
        ps.SetParameterString(1, "row 11");
        ps.ExecuteStatement();
        ps.SetParameterInt32(0, 37);
        ps.SetParameterString(1, "row 37");
        ps.ExecuteStatement();
        conn.Commit();
        ps.CloseObject();
        ps = conn.PrepareStatement("SELECT * FROM T");
        IAsyncOperation<ResultSet> op = ps.ExecuteQueryAsync();
        ResultSet rs = await op;
        int i;
        string str;
        while (rs.Next()) {
            i = rs.GetInt32(0);
            str = rs.GetString(1);
            // do something with row values
        }
        rs.CloseObject();
        ps.CloseObject();
    }
    catch (Exception ex) {
        // EXCEPTION HRESULT: ex.HResult
        if (ex.HResult == (int)ErrorCodes.E_ULTRALITE_ERROR) {
            if (conn != null) {
                conn.GetLastError(out code, out errorParms);
            } else {
                DatabaseManager.GetLastError(out code, out errorParms);
            }
            // do something with SQL code and error parameters
        }
    }
    finally {
        if (conn != null) {
            conn.CloseObject();
        }
    }
}
}

```

## Related Information

[Deploying an UltraLite Application for Windows Phone or Windows Store Apps \[page 769\]](#)



## 5.2.4 Deploying an UltraLite Application for Windows Phone or Windows Store Apps

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, link libraries, method calls, and deployment files to ensure that your UltraLite application runs successfully on Windows Phone devices or Windows Store Apps.

### Procedure

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

Synchronization type	Parameter settings
TCP/IP	Set the <i>Stream</i> synchronization parameter to tcpip.
HTTP	Set the <i>Stream</i> synchronization parameter to http.
RSA HTTPS	Set the <i>Stream</i> synchronization parameter to https.

3. When using ZLIB compression, set the protocol option `compression=zlib`.
4. Add a reference to the `UltraLite.winmd` file to the UltraLite WinRT component in your Microsoft Visual Studio project.

The `UltraLite.dll` library gets packaged with the application when a reference to `UltraLite.winmd` is added.

The WinRT components for the ARM, x86, and x64 processors are located in the `UltraLite\UWP\` directory of your SQL Anywhere installation. The Windows Phone 8 emulator for x86 processors is included in the respective directory.

### Results

The UltraLite application runs successfully on the Windows Phone device or Windows Store App that it is deployed to.

### Next Steps

Deploy an UltraLite database to the Windows Phone device or Windows Store App that the application was deployed to, or create a new database with the deployed application.

## Related Information

[UltraLite Database Deployment Techniques \[page 130\]](#)

[UltraLite Application Build and Deployment Specifications \[page 124\]](#)

## 5.3 Tutorial: Building a Windows Phone Application

Develop an application for Windows Phone by using the UltraLite for UWP API and the Microsoft Visual Studio environment. This tutorial assumes that the application runs on a Windows Phone emulator.

### Prerequisites

You need the following:

- Familiarity with C#
- Familiarity with Microsoft Visual Studio

You require the following software:

- Microsoft Visual Studio 2017

### Context

The Windows Phone application used in this tutorial is located in the `%SQLANY17%\UltraLite\UWP\WindowsPhone\CustDb\` directory. The application code, located in the `\CustDb` directory, references the UltraLite for UWP API to perform the following tasks:

- UltraLite remote database creation.
- SQL operations on the database.
- Data synchronization with the CustDB sample database by using MobiLink.

The XAML files illustrate how to create Windows Phone menu items and interfaces. View these files in Microsoft Visual Studio by opening the `CustDb.sln` solution file.

The Universal Windows Platform application's package manifest file was modified to permit network access, and to allow the application to generate a good quality GUID. The following Capability statements were added:

```
<Capability Name="ID_CAP_NETWORKING" />
<Capability Name="ID_CAP_IDENTITY_DEVICE" />
```

#### In this section:

[Lesson 1: Setting up a New Windows Phone Application \[page 771\]](#)

Create a new Windows Phone solution by using Microsoft Visual Studio.

[Lesson 2: Starting the MobiLink Server \[page 772\]](#)

Start the MobiLink server to perform synchronization.

[Lesson 3: Running Your Windows Phone Application and Synchronizing \[page 773\]](#)

Use your Windows Phone application to update the UltraLite remote database and synchronize the CustDB consolidated database.

[Lesson 4: Cleaning up \[page 774\]](#)

Remove tutorial materials from your computer.

## 5.3.1 Lesson 1: Setting up a New Windows Phone Application

Create a new Windows Phone solution by using Microsoft Visual Studio.

### Context

This tutorial assumes that you are familiar with C# and Microsoft Visual Studio.

### Procedure

1. Open `%SQLANYSAMPI7%\UltraLite.WinRT\WindowsPhone\CustDB\CustDb.sln` in Microsoft Visual Studio 2013.
2. Build the project. Click **Build** > **Build Solution**.

The project builds successfully.

### Results

The UltraLite for WinRT API is functional in the new Windows Phone application.

### Next Steps

Proceed to the next lesson.

## 5.3.2 Lesson 2: Starting the MobiLink Server

Start the MobiLink server to perform synchronization.

### Prerequisites

You must have completed the previous lessons in this tutorial.

### Procedure

1. Start MobiLink. Click **Start** > *SQLAnywhere 17* > *MobiLink* > *Synchronization Server Sample*.
2. Allow the Windows Phone application in the emulator to connect to the MobiLink server. Open the appropriate incoming port in the Windows firewall. For TCP/IP, the default is 2439, which is the IANA-registered port number for the MobiLink server.
  - a. Click **Start** > *Control Panel* > *Windows Firewall*.
  - b. Click *Advanced settings*.
  - c. Click *Inbound Rules*.
  - d. Click *New Rule*.
  - e. Click *Port*. Click *Next*.
  - f. Ensure that *TCP* is selected. Select *Specific local ports*. Type **2439**. Click *Next*.
  - g. Select *Allow the connection*. Click *Next*.
  - h. In the *Name* field, type **MobiLink Incoming**. Click *Finish*.

### Results

The MobiLink server starts. TCP port 2439 is open for incoming traffic.

### Next Steps

Proceed to the next lesson.

## 5.3.3 Lesson 3: Running Your Windows Phone Application and Synchronizing

Use your Windows Phone application to update the UltraLite remote database and synchronize the CustDB consolidated database.

### Prerequisites

You must have completed the previous lessons in this tutorial.

### Procedure

1. When the CustDb application runs for the first time, a configuration screen appears.
  - *Employee ID* identifies the employee using the application. The CustDb application shows only data relevant to the employee. The default is 50.
  - *Host* identifies the computer running the MobiLink server. If you are connected to the MobiLink server on a local area network, then you can use the computer's local IP address or host name. You can run `ipconfig` on the command line to find your computer's IP address.
  - *Port* refers to the incoming port number used by the MobiLink server. The default is 2439.
2. In the *Orders* screen, click *Sync*.

The application automatically synchronizes, and a set of customers, products, and orders is downloaded to the application from the CustDB consolidated database. Browse the orders by flicking up or down and tapping the individual order to bring up the *Order Details* screen.

3. Add a new order. Click *New*. Complete the form as indicated with the following information:

Option	Description
Customer	Ace Properties
Product	4x8 Drywall x100
Quantity	999
Discount	25

4. Click *OK* to add the new order.
5. Click *Sync* to synchronize the application with the CustDb consolidated database.
6. Confirm that the synchronization was successful. You can use Interactive SQL to connect to the CustDb consolidated database.
  - a. Click **Start** > *SQL Anywhere 17* > *Administration Tools* > *Interactive SQL*.
  - b. Click *ODBC Data Source Name* and choose *SQL Anywhere 17 CustDb*.
  - c. Click *Connect*.
7. Verify that the synchronization was successful.

Execute the following statement:

```
SELECT order_id, disc, quant, notes, status, c.cust_id,
       cust_name, p.prod_id, prod_name, price
FROM ULOrder o, ULCustomer c, ULProduct p
WHERE o.cust_id = c.cust_id
AND o.prod_id = p.prod_id
AND c.cust_name = 'Ace Properties'
AND p.prod_name = '4x8 Drywall x100'
```

Synchronization was successful when an order entry appears in Interactive SQL.

8. Close the emulator window.

## Results

The changes you made in the emulator are synchronized with the CustDB consolidated database.

## Next Steps

Proceed to the next lesson.

## 5.3.4 Lesson 4: Cleaning up

Remove tutorial materials from your computer.

## Prerequisites

You must have completed the previous lessons in this tutorial.

## Procedure

1. Close Microsoft Visual Studio.
2. Close MobiLink, Interactive SQL, and the synchronization client window by right-clicking each task bar item and clicking *Exit* or *Shut Down*.
3. Reset the CustDb database.

Run the following command from the `%SQLANYSAMPI7%\UltraLite\CustDb` directory:

```
makedbs
```

## Results

The materials are removed from your computer, and this tutorial can be repeated again from the first lesson.

## 5.4 UltraLite for UWP API Reference

UltraLite for UWP has a variety of API objects that you use from C#, C++, or JavaScript.

The following list describes some of the commonly used API objects:

### **DatabaseManager**

Provides methods for managing databases and connections.

### **Connection**

Represents a connection to an UltraLite database. You can create one or more Connection objects.

### **PreparedStatement, ResultSet**

Create dynamic SQL statements, make queries, execute INSERT, UPDATE, and DELETE statements, and attain programmatic control over database result sets.

## Namespace

```
UltraLite
```



The UltraLite UWP API reference is available in the *UltraLite - UWP API Reference* at <https://help.sap.com/viewer/4aa0fa9335514000836c47c2bec0bdc1/LATEST/en-US>.

# Important Disclaimers and Legal Information

## Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

## Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

## Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

## Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

## Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.





© 2022 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.