UltraLite®
Database Management and Reference

Contents

About this book

This book introduces the UltraLite database system for small devices.

About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in four formats:

- **DocCommentXchange** DocCommentXchange is a community for accessing and discussing SQL Anywhere documentation on the web.

  To access the documentation, go to http://dcx.sybase.com.

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

  To access the documentation, choose **Start** » **Programs** » **SQL Anywhere 12** » **Documentation** » **HTML Help (English)**.

- **Eclipse** On Unix platforms, the complete Help is provided in Eclipse format. To access the documentation, run *sadoc* from the *bin32* or *bin64* directory of your SQL Anywhere installation.

- **PDF** The complete set of SQL Anywhere books is provided as a set of Portable Document Format (PDF) files. You must have a PDF reader to view information.

  To access the PDF documentation on Windows operating systems, choose **Start** » **Programs** » **SQL Anywhere 12** » **Documentation** » **PDF (English)**.

  To access the PDF documentation on Unix operating systems, use a web browser to open */documentation/en/pdf/index.html* under the SQL Anywhere installation directory.

Documentation conventions

This section lists the conventions used in this documentation.

Operating systems

SQL Anywhere runs on a variety of platforms. Typically, the behavior of the software is the same on all platforms, but there are variations or limitations. These are commonly based on the underlying operating system (Windows, Unix), and seldom on the particular variant (IBM AIX, Windows Mobile) or version.

To simplify references to operating systems, the documentation groups the supported operating systems as follows:

- **Windows** The Microsoft Windows family includes platforms that are used primarily on server, desktop, and laptop computers, as well as platforms used on mobile devices. Unless otherwise

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

Windows Mobile is based on the Windows CE operating system, which is also used to build a variety of platforms other than Windows Mobile. Unless otherwise specified, when the documentation refers to Windows Mobile, it refers to all supported platforms built using Windows CE.

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

For the complete list of platforms supported by SQL Anywhere, see "Supported platforms" [*SQL Anywhere 12 - Introduction*].

Directory and file names

Usually references to directory and file names are similar on all supported platforms, with simple transformations between the various forms. In these cases, Windows conventions are used. Where the details are more complex, the documentation shows all relevant forms.

These are the conventions used to simplify the documentation of directory and file names:

- **Uppercase and lowercase directory names** On Windows and Unix, directory and file names may contain uppercase and lowercase letters. When directories and files are created, the file system preserves letter case.

  On Windows, references to directories and files are *not* case sensitive. Mixed case directory and file names are common, but it is common to refer to them using all lowercase letters. The SQL Anywhere installation contains directories such as *Bin32* and *Documentation*.

  On Unix, references to directories and files *are* case sensitive. Mixed case directory and file names are not common. Most use all lowercase letters. The SQL Anywhere installation contains directories such as *bin32* and *documentation*.

  The documentation uses the Windows forms of directory names. You can usually convert a mixed case directory name to lowercase for the equivalent directory name on Unix.

- **Slashes separating directory and file names** The documentation uses backslashes as the directory separator. For example, the PDF form of the documentation is found in *install-dir* *\Documentation\en\PDF* (Windows form).

  On Unix, replace the backslash with the forward slash. The PDF documentation is found in *install-dir/documentation/en/pdf*.

- **Executable files** The documentation shows executable file names using Windows conventions, with a suffix such as *.exe* or *.bat*. On Unix, executable file names have no suffix.

  For example, on Windows, the network database server is *dbsrv12.exe*. On Unix, it is *dbsrv12*.

- ***install-dir*** During the installation process, you choose where to install SQL Anywhere. The environment variable SQLANY12 is created and refers to this location. The documentation refers to this location as *install-dir*.

For example, the documentation may refer to the file *install-dir/readme.txt*. On Windows, this is equivalent to *%SQLANY12%\readme.txt*. On Unix, this is equivalent to *$SQLANY12/readme.txt* or *${SQLANY12}/readme.txt*.

For more information about the default location of *install-dir*, see "SQLANY12 environment variable" [*SQL Anywhere Server - Database Administration*].

- ***samples-dir*** During the installation process, you choose where to install the samples included with SQL Anywhere. The environment variable SQLANYSAMP12 is created and refers to this location. The documentation refers to this location as *samples-dir*.

  To open a Windows Explorer window in *samples-dir*, choose **Start** » **Programs** » **SQL Anywhere 12** » **Sample Applications And Projects**.

  For more information about the default location of *samples-dir*, see "SQLANYSAMP12 environment variable" [*SQL Anywhere Server - Database Administration*].

Command prompts and command shell syntax

Most operating systems provide one or more methods of entering commands and parameters using a command shell or command prompt. Windows command prompts include Command Prompt (DOS prompt) and 4NT. Unix command shells include Korn shell and bash. Each shell has features that extend its capabilities beyond simple commands. These features are driven by special characters. The special characters and features vary from one shell to another. Incorrect use of these special characters often results in syntax errors or unexpected behavior.

The documentation provides command line examples in a generic form. If these examples contain characters that the shell considers special, the command may require modification for the specific shell. The modifications are beyond the scope of this documentation, but generally, use quotes around the parameters containing those characters or use an escape character before the special characters.

These are some examples of command line syntax that may vary between platforms:

- **Parentheses and curly braces** Some command line options require a parameter that accepts detailed value specifications in a list. The list is usually enclosed with parentheses or curly braces. The documentation uses parentheses. For example:

  ```
  -x tcpip(host=127.0.0.1)
  ```

  Where parentheses cause syntax problems, substitute curly braces:

  ```
  -x tcpip{host=127.0.0.1}
  ```

  If both forms result in syntax problems, the entire parameter should be enclosed in quotes as required by the shell:

  ```
  -x "tcpip(host=127.0.0.1)"
  ```

- **Semicolons** On Unix, semicolons should be enclosed in quotes.

- **Quotes** If you must specify quotes in a parameter value, the quotes may conflict with the traditional use of quotes to enclose the parameter. For example, to specify an encryption key whose

value contains double-quotes, you might have to enclose the key in quotes and then escape the embedded quote:

```
-ek "my \"secret\" key"
```

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

- **Environment variables** The documentation refers to setting environment variables. In Windows shells, environment variables are specified using the syntax *%ENVVAR%*. In Unix shells, environment variables are specified using the syntax *$ENVVAR* or *${ENVVAR}*.

Contacting the documentation team

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

You can leave comments directly on help topics using DocCommentXchange. DocCommentXchange (DCX) is a community for accessing and discussing SQL Anywhere documentation. Use DocCommentXchange to:

- View documentation
- Check for clarifications users have made to sections of documentation
- Provide suggestions and corrections to improve documentation for all users in future releases

Go to http://dcx.sybase.com.

Finding out more and requesting technical support

Newsgroups

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

When you write to one of these newsgroups, always provide details about your problem, including the build number of your version of SQL Anywhere. You can find this information by running the following command: **dbeng12 -v**.

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

The newsgroups include the following:

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

For web development issues, see http://groups.google.com/group/sql-anywhere-web-development.

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

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

Developer Centers

The **SQL Anywhere Tech Corner** gives developers easy access to product technical documentation. You can browse technical white papers, FAQs, tech notes, downloads, techcasts and more to find answers to your questions as well as solutions to many common issues. See http://www.sybase.com/developer/library/sql-anywhere-techcorner.

The following table contains a list of the developer centers available for use on the SQL Anywhere Tech Corner:

Name	URL	Description
SQL Anywhere .NET Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/microsoft-net	Get started and get answers to specific questions regarding SQL Anywhere and .NET development.
PHP Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/php	An introduction to using the PHP (PHP Hypertext Preprocessor) scripting language to query your SQL Anywhere database.

Name	URL	Description
SQL Anywhere Windows Mobile Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/windows-mobile	Get started and get answers to specific questions regarding SQL Anywhere and Windows Mobile development.

Introducing UltraLite

UltraLite is a compact relational database with many of the same features as SQL Anywhere. It can be used as an embedded, in-process database for mobile applications.

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

See also

- "Understanding MobiLink synchronization" [*MobiLink - Getting Started*]
- "MobiLink clients" [*MobiLink - Client Administration*]
- "MobiLink - Getting Started"
- "MobiLink - Client Administration"
- "MobiLink - Server Administration"

UltraLite feature comparison

In the C/C++ version, the UltraLite database and management system adds 400-500 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, referential integrity, and multi-table joins	X	X	
Triggers, stored procedures, and views	X		

Feature	SQL Anywhere	UltraLite	Considerations
External stored procedures (callable external DLLs)	X		
Built-in referential and entity integrity	X	X	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. UltraLiteJ does not enforce foreign key constraints. See "Avoiding synchronization issues with foreign key cycles" on page 104.
Cascading updates and deletes	X	Limited	
Dynamic, multiple database support	X	X	With the UltraLite engine only.
Multi-threaded application support	X	X	
Row-level locking	X	X	
XML unload and load utilities		X	UltraLite uses separate administration tools to complete XML load and unloads. It is not built into the runtime. See "UltraLite Load XML to Database utility (ulload)" on page 205 and "UltraLite Database Unload utility (ulunload)" on page 214.
XML export and import utilities	X		SQL Anywhere uses SQL statements to export/import data to XML. You can also use dbunload to export your data. See "Importing and exporting data" [*SQL Anywhere Server - SQL Usage*].

Feature	SQL Anywhere	UltraLite	Considerations
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 a `Feature not available in UltraLite` error. See "UltraLite SQL functions" on page 266.
SQL statements	X	Limited	The scope of SQL statements are limited in UltraLite compared to SQL Anywhere. See "UltraLite SQL statements" on page 366.
Integrated HTTP server	X		
Strong encryption for database files and network communications	X	X	
Event scheduling and handling	X	X	UltraLite event model differs from SQL Anywhere. UltraLiteJ does not support events.
High-performance, self-tuning, cost-based query optimizer	X		UltraLite has a query optimizer, but it is not as extensive as that of SQL Anywhere. Therefore, the UltraLite optimizer may not provide as high performance as the SQL Anywhere optimizer on complex queries.
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. See "Choosing an UltraLite programming interface" on page 20.

Feature	SQL Anywhere	UltraLite	Considerations
Cursor support	X	X	See "UltraLite limitations" on page 7.
Dynamic cache sizing with an advanced cache management system	X		Cache sizing is static in UltraLite. Nonetheless, UltraLite allows you to set the cache size when the database is started, which gives you the ability to scale cache size. See "UltraLite CACHE_SIZE connection parameter" on page 167.
Database recovery after system or application failure	X	X	
Binary Large Object (BLOB) support	X	X	UltraLite cannot index or compare BLOBs.
Windows Performance Monitor integration	X		
Online table and index defragmentation	X		
Online backup	X		

Feature	SQL Anywhere	UltraLite	Considerations
Small footprint, which can be as small as 500 KB		X	Small footprint devices tend to have relatively slow processors. UltraLite employs algorithms and data structures that are targeted for these devices, so UltraLite continues to provide high performance and low memory use.
Direct device connections to a 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 Windows Mobile device. On UltraLite, you simply need to prefix the connection string with **WCE:**. See "Windows Mobile" on page 37.
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. See "UltraLite max_hash_size creation parameter" on page 143.
Synchronizing to Oracle, DB2, Sybase Adaptive Server Enterprise, Microsoft SQL Server, My SQL Sybase, or SQL Anywhere	X	X	

Feature	SQL Anywhere	UltraLite	Considerations
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 need to deploy. See "UltraLite clients" on page 93.
In-process execution		X	
Computed columns	X		
Declared temporary tables/ global temporary tables	X		
System functions	X		UltraLite does not support SQL Anywhere system functions, including property functions. You cannot include them as part of your UltraLite application.
Timestamp columns	X	X	SQL Anywhere Transact-SQL timestamp columns are created with the DEFAULT TIMESTAMP default. UltraLite timestamp columns are created with the DEFAULT CURRENT TIMESTAMP default. Therefore, UltraLite does 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. See "UltraLite user authentication" on page 39.
Spatial data	X	Limited	UltraLite and UltraLiteJ both support point data only.
Full text data	X		

UltraLite limitations

To compare UltraLite limitations with SQL Anywhere limitations, see "SQL Anywhere size and number limitations" [*SQL Anywhere Server - Database Administration*].

Statistic	Maximum for UltraLite
Number of connections per database	Up to 14 for single threaded applications.
Number of concurrent open connections	Up to 32 for all OS.
Total number of concurrent connections per application	Up to 64 for all OS.
SQL communication areas (SQLCA)	Up to 63.
File-based persistent store (database size)	2 GB file or OS limit on file size.
Rows per table	Up to 16 million.[1]
Row Size	The length of each packed row must not exceed the page size. See "Row packing and table definitions" on page 53. 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 are stored separately
Rows per database	Limited by persistent store.
Table size	Limited only by database size.
Tables per database	Limited only by 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 only by database size.
Tables referenced per transaction	No limit.

Statistic	Maximum for UltraLite
Stored procedure length	Not applicable.
Stored procedures per database	Not applicable.
Triggers per database	Not applicable.
Nesting	Not applicable.
Number of publications	Up to 63.
Database page size	16 KB.
Cursors per connection	The number of maximum allowable cursors on a given connection to an UltraLite database is 64 (all platforms).
Long binary/long varchar size	Limited only by database size.
UltraLiteJ Compatibility	UltraLite databases are not interchangeable with UltraLiteJ databases.

[1]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, or whether the table is synchronized at all. See "UltraLite transaction processing" on page 14.

UltraLite data architecture

UltraLite is a mobile database designed to create custom solutions for small-footprint devices such as cell phones, handheld computers, and embedded devices.

UltraLite provides you with a complete database management system including:

- **Development layer** UltraLite supports several programming interfaces that keep you from getting locked into one deployment platform, development tool, or set of IT infrastructure products.

  For information about which API you should choose, see "Choosing an UltraLite programming interface" on page 20.

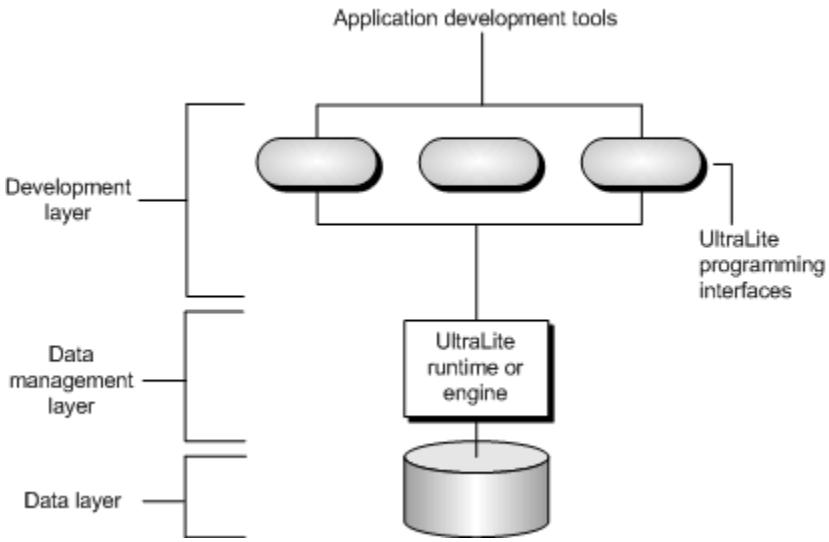
  To help you maintain your UltraLite project, UltraLite completes its development support with a comprehensive set of administration tools. You can run these tools either as command line utilities or as wizards in the UltraLite plug-in for Sybase Central.

- **Data management layer and synchronization client** You can connect to an UltraLite database with an in-process runtime library or a separate process called an engine. Both processes control connection and data access requests. They also include a built-in bi-directional synchronization client that links UltraLite databases with enterprise back-end systems.

  For information about which process you should choose, see "Choosing an UltraLite data management component" on page 19.

- **Data layer** This layer is the local data repository stored as a file. See "UltraLite storage and file name conventions" on page 9.

The data, management, and development layers are represented in the following figure.



UltraLite storage and file name conventions

The file management requirements of a device dictate how an UltraLite database is stored and what database name conventions must be used. Depending on platform restrictions, you may be able to create a database on the development desktop, and then deploy it to one or more platforms.

See also

- "Specifying file paths in an UltraLite connection parameter" on page 36

UltraLite database schema

The logical framework of the database is known as a **schema**. In UltraLite, the schema is maintained as a catalog of system tables that hold the metadata for the UltraLite database. Metadata stored in the system tables includes:

- Index definitions. See "sysindex system table" on page 221 and "sysixcol system table" on page 222.
- Table definitions. See "systable system table" on page 220.
- Column definitions. See "syscolumn system table" on page 220.
- Publication definitions. See "syspublication system table" on page 223 and "sysarticle system table" on page 223.
- User names and passwords. See "sysuldata system table" on page 224.

Schema changes with DDL statements

You can change the schema of a 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 (for example, 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.

See also

- "ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]" on page 368
- "UltraLite SQL statements" on page 366

UltraLite temporary files

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 has a few differences.

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 this file.

For record-based platforms you can setup UltraLite to save the temporary file in another location by supplying the TEMP_DIR connections parameter. See "UltraLite TEMP_DIR connection parameter" on page 183

UltraLite 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. 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.

You can avoid using temporary tables by using an index for the columns used in the ORDER BY or GROUP BY clauses.

See also

- "UltraLite temporary files" on page 10
- "UltraLite performance and optimization" on page 81
- "When to view an execution plan" on page 263

UltraLite transaction and 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. See "Concurrency in UltraLite" on page 11.
- Synchronization progress state, to ensure that synchronization occurs successfully. See "Built-in UltraLite synchronization features" on page 93.
- Row state, to maintain data integrity by tracking how data has changed between synchronizations. See "UltraLite row states" on page 12.
- Transactions, to determine when and how data gets committed. In UltraLite, a transaction is processed in its entirety or not at all. See "UltraLite transaction processing" on page 14.
- 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. See "Backing up and recovering data in UltraLite" on page 14.

Concurrency in UltraLite

UltraLite uses the following methods to manage concurrent database accesses.

- **Multiple databases** A single UltraLite application can open connections to up to 32 databases.
- **Multiple applications** The UltraLite database can only be opened by one process at a time. If you plan to support concurrency among multiple applications, choose the UltraLite engine as your data management component. See "Choosing an UltraLite data management component" on page 19.
- **Multiple threads** UltraLite supports multi-threaded applications. A single application can be written to use multiple threads, each of which can connect to the same or different databases.

  If you are managing your database with the runtime there is a limit of 64 concurrent connections.

  If you are managing your database connections with the UltraLite engine, the number of SQLCAs you can use is typically restricted to 128. However, the implementation of UltraLite.NET API effectively reduces this limit to 128 less the number of running UltraLite.NET clients.
- **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. See "UltraLite transaction processing" on page 14.
- **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. See "Additional Parameters synchronization parameter" on page 111.

  If synchronization fails, UltraLite supports resumable downloads on all platforms. See "Handling failed downloads" [*MobiLink - Server Administration*].

See also

- "UltraLite clients" on page 93
- "Additional Parameters synchronization parameter" on page 111

UltraLite row states

Maintaining row state information is a powerful part of the UltraLite feature set. Tracking the state of tables and rows is particularly important for data synchronization.

Changes to data

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. Restore the original state of the row to undo the delete.
- **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.

See also

- "Backing up and recovering data in UltraLite" on page 14
- "Flushing single or grouped transactions" on page 89
- "UltraLite transaction processing" on page 14

Validate an UltraLite database

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

- The **Validate Database Wizard** in Sybase Central.
- Calling the ValidateDatabase function or method depending on your API.
- The ulvalid command line utility.

UltraLiteJ does not support database validation.

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.

To validate a database (Sybase Central)

1. In the left pane of Sybase Central, select the UltraLite database.
2. Choose **File** » **Validate Database**.
3. Follow the instructions in the **Validate Database Wizard**.

See also

- .NET: "ValidateDatabase method" [*UltraLite - .NET Programming*]
- C++: "ValidateDatabase method" [*UltraLite - C and C++ Programming*]
- "UltraLite Validate Database utility (ulvalid)" on page 218

Backing up and recovering data in UltraLite

If an application using an UltraLite database stops unexpectedly, the UltraLite 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 UltraLite database. All transactions not flushed at the time of the failure are rolled back.

UltraLite 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. See "UltraLite row states" on page 12.

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.

In smaller UltraLite deployments, you can copy the database file to a desktop computer to provide a manual backup.

See also

- "Flushing single or grouped transactions" on page 89

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.

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.

For more information about how to handle errors, see:

- UltraLite for C/C++: "Handling errors" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "Handling errors" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "Handling errors" [*UltraLite - M-Business Anywhere Programming*]

Programming tip
All UltraLite APIs—except the C++ API—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.

See:

- UltraLite.NET: "Managing transactions" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "Managing transactions" [*UltraLite - M-Business Anywhere Programming*]

UltraLite 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 in auto-commit mode. For .NET, read-committed is the default isolation level for new transactions created by calling ULConnection.BeginTransaction without parameters. The default UltraLite isolation level provides the best performance while ensuring data consistency.

With the ReadCommitted isolation level:

- Dirty reads are prevented
- No read locks are applied
- Uncommitted deletes are visible
- Non-repeatable reads and phantom rows are allowed
- No guarantee that data will not change during transaction

With the ReadUncommitted isolation level:

- Dirty reads are allowed
- No read locks are applied
- Non-repeatable reads and phantom rows are allowed
- No guarantee that concurrent transaction will not modify row or roll back changes to row

You can change the isolation level from ReadCommitted to ReadUncommitted. For UltraLite C++, use the SetDatabaseOption method to change the isolation level. For UltraLite.NET 2.0, call the ULConnection.BeginTransaction to create a transaction with the ReadUncommitted isolation level. UltraLiteJ only supports the ReadUncommitted isolation level.

Note
For UltraLite.NET, executing SetDatabaseOption while a transaction is active is not recommended. It changes the isolation level of the connection, but does not update the ULTransaction.IsolationLevel.

Do not use SetDatabaseOption to change the isolation while a transaction is in progress; unpredictable results might occur.

Isolation level side effects

As UltraLite operates by default at an isolation level of 0, which is also known as ReadUncommitted the following side effects are possible:

- No locking operations are required when executing a SELECT statement.
- Applications can read uncommitted data (also known as dirty reads). In this scenario, transaction 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).

  For a tutorial that demonstrates the effects of dirty reads, see "Tutorial: Dirty reads" [*SQL Anywhere Server - SQL Usage*]. For a tutorial that demonstrates a phantom row, see "Tutorial: Phantom rows" [*SQL Anywhere Server - SQL Usage*].
- 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.

  For a tutorial that demonstrates the effects of non-repeatable reads, see "Tutorial: Non-repeatable reads" [*SQL Anywhere Server - SQL Usage*].

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.

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[1]
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.

[1] Queries used by Connection A and B do not contain temporary tables. Temporary tables can cause other side effects.

See also

- "BeginTransaction method" [*UltraLite - .NET Programming*]
- "SetDatabaseOption method" [*UltraLite - .NET Programming*]

Implementing an UltraLite solution

When implementing an UltraLite solution, consider the following:

- How many applications need to connect to the UltraLite database? The number of concurrent connections affects whether you need the UltraLite in-process runtime or the UltraLite engine. To understand how they differ, see "Choosing an UltraLite data management component" on page 19.
- What platform(s) do you want to support? This choice can affect which APIs are available to program your application. See "Choosing an UltraLite programming interface" on page 20.
- Which platforms will the database run on? Because file formats have been consolidated, you may be able to create a database that runs on multiple platforms. See "Creating and configuring UltraLite databases" on page 23.

> **Tip**
> If you need to create a file format that suits multiple platforms, use the **Create Database Wizard** in Sybase Central to help you determine whether a single database is possible.

UltraLite supported platforms and protocols

You can synchronize the data in UltraLite databases with a central consolidated database over the TCP/IP, HTTP, or HTTPS network protocols.

For more information about the device platforms and different network protocols (also known as streams) supported by UltraLiteJ, see http://www.sybase.com/detail?id=1061806.

See also

- "UltraLite clients" on page 93
- "Network protocol options for UltraLite synchronization streams" on page 133
- "UltraLite Synchronization utility (ulsync)" on page 209

Choosing an UltraLite data management component

UltraLite allows you to build a small-footprint relational database solution without requiring the additional overhead of setting up a separate database server. Instead, UltraLite programming interfaces use one of two library types.

UltraLite in-process runtime library

In UltraLite, the runtime and the application are part of the same process, which makes the database specific to the application. For all platforms, the runtime manages UltraLite databases and built-in synchronization operations. The UltraLite runtime can manage a maximum of 14 databases at any one time. Note that only one application can access a database at any given time.

On Windows or Windows Mobile, linking to the runtime requires that you use a specific import library/DLL pair than that of the engine. For details, see "Compiling and linking your application" [*UltraLite - C and C++ Programming*].

If you require TLS-enabled synchronization, there are additional libraries you also require. See "Deploy UltraLite with TLS-enabled synchronization" on page 47.

On the iPhone, UltraLite is available only as an in-process runtime library. For information about building an UltraLite application using the iPhone SDK See "UltraLite for C/C++ developers" [*UltraLite - C and C++ Programming*].

UltraLiteJ for BlackBerry and Java SE is provided as an in-process JAR file. For information about building an UltraLiteJ application using the BlackBerry JDE, see "Developing UltraLiteJ applications" [*UltraLiteJ*].

UltraLite database engine (uleng12.exe)

The UltraLite engine is only available for Windows desktop, Windows Mobile, and Linux platforms. The engine is a separate executable, which 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 a different import library/DLL pair than that of the runtime.

Additional libraries

If you require TLS-enabled synchronization or AES FIPS database encryption, there are additional libraries you also require. See "Deploy UltraLite with AES_FIPS database encryption" on page 46 and "Deploy UltraLite with TLS-enabled synchronization" on page 47.

See also

- "UltraLite Engine utility (uleng12)" on page 194
- "Concurrency in UltraLite" on page 11
- "UltraLite feature comparison" on page 1
- "UltraLite limitations" on page 7

Choosing an UltraLite programming interface

UltraLite APIs offer different 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 the creation of applications for your varied deployment environments.

To choose your programming interface

1. Choose your target platform(s).

For each platform you need to support, determine if the API supports that platform. Different APIs support different platforms. If you are doing cross-platform development, choose an API that supports all of your intended targets.

The support matrix found here: http://www.sybase.com/detail?id=1061806, can be used to identify your development options.

2. Consider the effects of the following requirements, and then finalize your selection:

   SQL Anywhere compatibility If database compatibility with SQL Anywhere is a concern, consider the following:

   - SQL Anywhere embedded SQL support provides a common programming interface for UltraLite and SQL Anywhere databases.
   - ADO.NET provides common programming models that are shared between UltraLite components and SQL Anywhere.

   Maintaining a common interface may be particularly useful on platforms such as Windows Mobile, where both SQL ANywhere and UltraLite databases are supported. If you need to move from UltraLite to a SQL Anywhere database, you should use embedded SQL or ADO.NET to make application migration easier.

   Simplified deployments If simplifying your UltraLite deployment is an issue, consider programming with the M-Business Anywhere API. Your end-users can download both the UltraLite application and the database concurrently.

   Application size If creating the smallest application footprint is a priority, you should program your application with the C/C++ API. These applications typically yield the best performance and still maintain a small application file size.

   Application performance Each API yields a different performance result. While UltraLite provides high performance in a variety of environments and use cases, embedded SQL and the C++ API are the lowest level of APIs and generally deliver the highest performance.

See also

- UltraLite.NET: “UltraLite - .NET Programming”
- UltraLite C/C++ and embedded SQL: “UltraLite - C and C++ Programming”
- UltraLite for M-Business Anywhere: “UltraLite - M-Business Anywhere Programming”

Using UltraLite databases

This part describes how to create and use UltraLite databases.

Creating and configuring UltraLite databases

There are three types of database creation methods:

- Desktop creation methods with UltraLite administration tools designed for this purpose.
- 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.

Sharing a database among multiple platforms

Within the configuration differences imposed by different operating systems, you might be able to copy the database from one device to another. If you are unsure of property compatibility among multiple platforms, create a database in Sybase Central with the **UltraLite Create Database Wizard**. This wizard handles the file compatibility logic for you. In so doing, it prevents you from creating a file that is not supported on your combination of deployment devices.

See also

- "Choosing database creation parameters for UltraLite" on page 28
- "Change UltraLite persistent database option settings" on page 167
- "Working with UltraLite databases" on page 52

Creating an UltraLite database

The UltraLite Create Database Wizard Choose this method if you want help navigating the available database creation parameters. 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. See "Create a database with the Create Database Wizard" on page 24.

The MobiLink Create Synchronization Model Wizard Choose this method if you are creating a synchronization system with remote UltraLite databases and a centralized consolidated database.

See "Create an UltraLite database from a MobiLink synchronization model" on page 24.

The command line Choose any of the following utilities:

- Use the ulinit utility if you want to create a new, empty UltraLite database or one sourced from a SQL Anywhere reference database schema. See "Create an UltraLite database from a SQL Anywhere reference database" on page 25.
- Use the ulload 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. See "Create an UltraLite database from XML" on page 26.
- 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. See "Central administration of remote databases" [*MobiLink - Server Administration*].

Create a database with the Create Database Wizard

You can create a database in Sybase Central using the **Create Database Wizard**.

To create a new UltraLite database (Sybase Central)

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. Choose **Tools** » **UltraLite 12** » **Create Database**.
3. Follow the instructions in the **Create Database Wizard**.

Create an UltraLite database from the command prompt

Use the ulinit utility to create a database from a command prompt. With this utility, you can include utility options to configure the database.

To create a new UltraLite database (command line)

Run the ulinit utility specifying the new UltraLite database file if you want to accept the defaults:

```
ulinit test.udb
```

Refer to "UltraLite Initialize Database utility (ulinit)" on page 197 for a description of all available options.

Create an UltraLite database from a MobiLink synchronization model

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

Once you have created your model, you can work in MobiLink Model mode in Sybase 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.

See "MobiLink Plug-in for Sybase Central" [*MobiLink - Getting Started*].

Create an UltraLite database using central administration of remote databases

MobiLink provides a create database command that allows you to create an UltraLite database. See "Create database command" [*MobiLink - Server Administration*] and "Central administration of remote databases" [*MobiLink - Server Administration*].

Create an UltraLite database from a SQL Anywhere reference database

A reference database is a SQL Anywhere database that serves as a template for the UltraLite database you are creating. Your UltraLite database is a subset of the columns, tables, and indexes in the reference database. You select these objects as part of a publication in the reference database. You can also choose to include data from your SQL Anywhere database in your new UltraLite database.

Creating a database from a reference database may be useful if you want to first model your data with an architecture tool such as Sybase PowerDesigner.

To create a database from a reference database, use the ulinit utility. See "Create an UltraLite database from a SQL Anywhere reference database" on page 25 and "UltraLite Initialize Database utility (ulinit)" on page 197.

To initialize/extract a new UltraLite database from a reference database (command line)

1. Create a new SQL Anywhere database as the reference database.

   You can create a new SQL Anywhere database with the dbinit utility or Sybase Central. You can also create a SQL Anywhere database from non-SQL Anywhere databases, by migrating data from these third-party files.

   See "Creating a SQL Anywhere database" [*SQL Anywhere Server - Database Administration*].

   Configure the database with UltraLite usage in mind The UltraLite database is generated with the same settings as those in the reference database. By setting the following options in the reference database, you also control the behavior of your UltraLite database:

- Date format (see "UltraLite date_format creation parameter" on page 138)
- Date order (see "UltraLite date_order creation parameter" on page 141)
- Nearest century (see "UltraLite nearest_century creation parameter" on page 144)
- Precision (see "UltraLite precision creation parameter" on page 148)
- Scale (see "UltraLite scale creation parameter" on page 149)
- Time format (see "UltraLite time_format creation parameter" on page 150)
- Timestamp format (see "UltraLite timestamp_format creation parameter" on page 152)
- Timestamp with timezone (see "UltraLite timestamp_with_time_zone_format creation parameter" on page 155)

2. Prepare the reference database by adding objects required by the UltraLite database:
   - **Tables and keys** Add the tables and remember to set primary keys as they are required by UltraLite. If you need to, you can also assign foreign key relationships that you need within your UltraLite application. You can use Sybase Central, Sybase PowerDesigner Physical Data Model, or another database design tool. See "Working with UltraLite tables and columns" on page 52.
   - **Indexes** An index can improve performance dramatically, particularly on slow devices. Note that primary keys are automatically indexed, but other columns are not. See "When to use an index" on page 62.
   - **Publications** If you want to synchronize different tables at different times, use publications. You can use multiple UltraLite publications to define table subsets and set synchronization priority with them. See "Publications in UltraLite" on page 102.

   Performance tip
   If your UltraLite applications frequently retrieve information in a particular order, consider adding an index to your reference database specifically for this purpose. See "Using index scans" on page 82.

3. Run the ulinit utility, including any necessary options:

   ```
   ulinit -a DBF=MySource.db customer.udb -n Pub1 -s Pub2
   ```

   In this example, MySource.db is the SQL Anywhere reference database and customer.udb is the UltraLite database that gets created. The tables and indexes will match those contained in the Pub1 and Pub2 publications. A publication for the UltraLite database is created for Pub2.

See also

- "Create a database with the Create Database Wizard" on page 24
- "Create an UltraLite database from the command prompt" on page 24
- "Create an UltraLite database from XML" on page 26
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "Choosing database creation parameters for UltraLite" on page 28
- "Upgrading UltraLite" [*SQL Anywhere 12 - Changes and Upgrading*]

Create an UltraLite database from XML

You can use XML as an intermediate format for managing your UltraLite database if the format follows the requirements for UltraLite usage. You can use XML as follows:

- 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.

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

To create an UltraLite database from an XML file

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. See "Exporting relational data as XML" [*SQL Anywhere Server - SQL Usage*].
   - 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.

   For example, to create a new UltraLite database in the file *sample.udb* from the table formats and data in *sample.xml*:

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

See also

- "Create a database with the Create Database Wizard" on page 24
- "Create an UltraLite database from the command prompt" on page 24
- "Create an UltraLite database from a SQL Anywhere reference database" on page 25
- "Upgrading UltraLite" [*SQL Anywhere 12 - Changes and Upgrading*]
- "UltraLite Load XML to Database utility (ulload)" on page 205
- "Choosing database creation parameters for UltraLite" on page 28

Create an UltraLite database 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.

See also

- "Creating an UltraLite database" on page 23
- UltraLite for C/C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]

Choosing database creation parameters for UltraLite

There are several options you can control when creating your UltraLite database. These are designed to help with the wide variety of UltraLite uses. Note that most parameters specified at creation time cannot be changed later.

Accessing creation parameter values

You cannot change creation parameter values after you have created a database. However, you can view the corresponding database properties in Sybase Central. See "Accessing UltraLite database properties" on page 162.

You can also access the database properties programmatically from the UltraLite application by calling the GetDatabaseProperty function appropriate to the API.

For API-specific details, see:

- UltraLite C/C++: "GetDatabaseProperty method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "GetDatabaseProperty method" [*UltraLite - .NET Programming*]
- M-Business: "getDatabaseProperty method" [*UltraLite - M-Business Anywhere Programming*]

In addition to these database creation parameters, you can further configure other aspects of your database with database options and connection parameters. See:

- "UltraLite database options" on page 162
- "Connecting to an UltraLite database" on page 34
- "UltraLite connection parameters" on page 167

Property	Description
case	Sets the case-sensitivity of string comparisons in the UltraLite database. See "UltraLite case creation parameter" on page 135.
checksum_level	Sets the level of checksum validation in the database. See "UltraLite checksum_level creation parameter" on page 136.

Property	Description
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. See "UltraLite character sets" on page 30 and "UltraLite collation creation parameter" on page 137 and "UltraLite utf8_encoding creation parameter" on page 157.
date_format	Sets the default string format in which dates are retrieved from the database. See "UltraLite date_format creation parameter" on page 138.
date_order	Controls the interpretation of date ordering of months, days, and years. See "UltraLite date_order creation parameter" on page 141.
fips	Controls AES FIPS compliant data encryption, by using a Certicom certified cryptographic algorithm. FIPS encoding is a form of strong encryption. See "Securing UltraLite databases" on page 32 and "UltraLite fips creation parameter" on page 142.
max_hash_size	Sets the default index hash size in bytes. See "UltraLite max_hash_size creation parameter" on page 143.
nearest_century	Controls the interpretation of two-digit years in string-to-date conversions. See "UltraLite nearest_century creation parameter" on page 144.
obfuscate	Controls whether data in the database is obfuscated. Obfuscation is a form of simple encryption. See "Securing UltraLite databases" on page 32 and "UltraLite obfuscate creation parameter" on page 146.
page_size	Defines the database page size. See "UltraLite page_size creation parameter" on page 146.
precision	Specifies the maximum number of digits in decimal point arithmetic results. See "UltraLite precision creation parameter" on page 148.
scale	Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision. See "UltraLite scale creation parameter" on page 149.
time_format	Sets the format for times retrieved from the database. See "UltraLite time_format creation parameter" on page 150.
timestamp_format	Sets the format for timestamps retrieved from the database. See "UltraLite timestamp_format creation parameter" on page 152.
timestamp_increment	Determines how the timestamp is truncated in UltraLite. See "UltraLite timestamp_increment creation parameter" on page 154.

Property	Description
time-stamp_with_time_zone_format	This option sets the format for TIMESTAMP WITH TIME ZONE values retrieved from the database. See "UltraLite timestamp_with_time_zone_format creation parameter" on page 155.
utf8_encoding	Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode. See "UltraLite character sets" on page 30 and "UltraLite utf8_encoding creation parameter" on page 157.

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 sort order you require. Generally speaking, you should 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.

See also

- "UltraLite platform requirements for character set encoding" on page 31
- "UltraLite collation creation parameter" on page 137
- "UltraLite utf8_encoding creation parameter" on page 157
- "Understanding character sets" [*SQL Anywhere Server - Database Administration*]
- "UltraLite connection parameters" on page 167
- "Character set considerations" [*MobiLink - Server Administration*]
- "UltraLite case creation parameter" on page 135
- "Securing UltraLite databases" on page 32

UltraLite platform requirements for character set encoding

Each platform has specific character set and encoding requirements.

Windows desktop and Windows Mobile

When using a UTF-8 encoded database on 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 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.

See also

- "UltraLite utf8_encoding creation parameter" on page 157
- "Understanding character sets" [*SQL Anywhere Server - Database Administration*]
- "UltraLite connection parameters" on page 167
- "Character set considerations" [*MobiLink - Server Administration*]
- "Securing UltraLite databases" on page 32

UltraLite supported collations

The following table lists the supported CHAR collations 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

Collation label	Description
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
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

Securing UltraLite databases

By default, UltraLite databases are unencrypted on disk. When using a viewing tool such as a hex editor, text and binary columns can be read. To encrypt data for greater security, consider the following options:

- **Obfuscation** Also known as simple encryption, this option provides protection against casual attempts to access data in the database. It does not provide as much security as strong encryption. Obfuscation has a minimal performance impact. You enable obfuscation with the obfuscate creation parameter. End users do not need to supply a corresponding connection parameter. You do not need

any special configuration to use simple obfuscation on your device. See "UltraLite obfuscate creation parameter" on page 146.

- **AES 256-bit strong encryption** UltraLite databases can be strongly encrypted using the AES 256-bit algorithm, which is the same algorithm used to encrypt SQL Anywhere databases. Strong encryption provides security against skilled and determined attempts to gain access to the data, but has a significant performance impact. You set encryption in the wizards by selecting the **Encrypt Database** option and then selecting **AES Strong Encryption**. Using a creation utility, you set the key with the key connection parameter. This same parameter is used by end users when connecting to the database after it has been created. You do not need any special configuration to use AES encryption on your device. See "UltraLite fips creation parameter" on page 142.

- **AES FIPS 140-2 compliant encryption** UltraLite provides encryption libraries compliant with the FIPS 140-2 US and Canadian government standard (using a Certicom certified cryptographic module). You set FIPS compliant encryption with the fips creation parameter. The user must supply the required key in their connection string. AES FIPS encryption requires that you configure your device appropriately. See "Deploy UltraLite with AES_FIPS database encryption" on page 46, and "UltraLite fips creation parameter" on page 142.

Note
Both the FIPS and AES database encryption types use 256-bit AES. Therefore, if you use the same encryption key, the database is encrypted the same way irrespective of the standard you choose.

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

- UltraLite for C++: "ChangeEncryptionKey method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "ChangeEncryptionKey method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "changeEncryptionKey method" [*UltraLite - M-Business Anywhere Programming*]

This operation is costly and is non-recoverable: if your operation terminates mid-course, you will lose your database entirely.

For strongly encrypted databases, be sure to 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.

See also

- "UltraLite fips creation parameter" on page 142
- "UltraLite obfuscate creation parameter" on page 146
- "UltraLite DBKEY connection parameter" on page 174
- "Deploy UltraLite with TLS-enabled synchronization" on page 47
- UltraLite.NET: "Encryption and obfuscation" [*UltraLite - .NET Programming*]
- UltraLite for C++: "Encrypting data" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Database encryption and obfuscation" [*UltraLite - M-Business Anywhere Programming*]

Connecting to an UltraLite database

Any application that uses a database must establish a connection to that database before any transactions can occur. An application can be an UltraLite command line utility, a connection window from either Sybase Central tool or Interactive SQL, or your own custom application.

By connecting to an UltraLite database, you form a channel through which all activity from the application takes place. Each connection attempt creates a database specific SQL transaction.

UltraLite database connection parameters

UltraLite supports the following connection parameters.

Parameter name	Description
CACHE_SIZE	Defines the size of the database cache. See "UltraLite CACHE_SIZE connection parameter" on page 167.
COMMIT_FLUSH	Determines when committed transactions are flushed to storage after a commit call. See "UltraLite COMMIT_FLUSH connection parameter" on page 170.
CON	Specifies a name of the current connection. See "UltraLite CON connection parameter" on page 171.
DBF, CE_FILE, desktop, device, and NT_FILE	At creation time these parameters set the location of the database. For subsequent connections, they tell UltraLite where to find the file. You can use DBF if you are creating a single-platform application or are connecting to an UltraLite administration tool. Use the other platform-specific versions if you are programming an UltraLite client that connects to different platform-specific databases. See: • "UltraLite DBF connection parameter" on page 172 • "UltraLite CE_FILE connection parameter" on page 169 • "UltraLite desktop connection parameter" on page 175 • "UltraLite device connection parameter" on page 177 • "UltraLite NT_FILE connection parameter" on page 179
DBN	Identifies a running database by name rather than file name. See "UltraLite DBN connection parameter" on page 174.
DBKEY	Specifies the encryption key used to encrypt the database. See "UltraLite DBKEY connection parameter" on page 174.

Parameter name	Description
MIRROR_FILE	Specifies the name of a database mirror file. See "UltraLite MIRROR_FILE connection parameter" on page 178.
PWD	At creation-time, sets the initial password for a user. For subsequent connections, supplies the password for the user ID. See "UltraLite PWD connection parameter" on page 180.
RESERVE_SIZE	Pre-allocates the file system space required for your UltraLite database without actually inserting any data. See "UltraLite RESERVE_SIZE connection parameter" on page 181.
START	Specifies the location of the UltraLite engine executable. See "UltraLite START connection parameter" on page 182.
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). See "UltraLite TEMP_DIR connection parameter" on page 183.
UID	At creation time, sets the initial user ID. For subsequent connections, identifies a user to the database. The user ID must be one of up to four user IDs stored in the UltraLite database. See "UltraLite UID connection parameter" on page 184.

See also

- "Interpreting user ID and password combinations" on page 40

Supplying UltraLite connection parameters

Connection details can be collected via different methods, depending on whether you are connecting from a custom UltraLite application or from one of the SQL Anywhere administration tools for UltraLite.

Method
Prompt the end user at connection time when you require a user to authenticate as one of the four supported database users. The UltraLite graphical administration tools use a connection object. You can use the UltraLite.Net ULConnectionParms object or the UltraLite for M-Business Anywhere ConnectionParms object to aid connection parameter collection and inspection. See: • UltraLite.NET: "ULConnectionParms class" [*UltraLite - .NET Programming*] and • UltraLite for M-Business Anywhere: "ConnectionParms class" [*UltraLite - M-Business Anywhere Programming*]

Method
Use a fixed connection string if user authentication is not required. Common reasons for not authenticating a user 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. The UltraLite command line utilities typically use a connection string if a connection to a database is required. You can also program your UltraLite application to read the values from a stored file, or hard code it into your application. See: • UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*] • UltraLite C/C++ : "Connecting to a database" [*UltraLite - C and C++ Programming*] • UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*] • UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
dbisql is the only tool that supports ULSQLCONNECT. Use the ULSQLCONNECT environment variable if you want to store connection parameters you use repeatedly. By storing parameters, you don't need to provide them repeatedly during the development phase. Values you supply as a parameter in ULSQLCONNECT become defaults for the UltraLite desktop administration tools.

[1] Typically user-supplied.

[2] For desktop administration tools only.

[3] Typically hard coded or stored in a file.

See also

- "Precedence of connection parameters for UltraLite administration tools" on page 38

Specifying file paths in an UltraLite connection parameter

The physical storage of your device determines:

- Whether the database is saved as a file.
- What naming conventions you must follow when identifying your database.

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
```

Windows Mobile tip
You can use the UltraLite administration tools to administer databases already deployed to an attached device. See "Windows Mobile" on page 37.

Otherwise, if you are writing a cross-platform application, use the platform specific (CE_DBF, or NT_DBF) file connection parameters to construct a universal connection string. For example:

```
Connection = DatabaseManager::OpenConnection("UID=JDoe;PWD=ULdb;CE_DBF=
\database\MyCEDB.udb;NT_FILE=MyDB.udb")
```

Desktop

Desktops allow either absolute or relative paths.

Windows Mobile

Windows Mobile devices require that all paths be absolute.

You can administer a Windows Mobile database on either the desktop or the attached device. To administer a database on a 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 Windows Mobile device to the *ce.xml* file in the Windows desktop folder of *c:\out*.

If you are using the ulunloadold or ulunload utilities to administer a database on the 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 utilities.

See also

- "UltraLite DBF connection parameter" on page 172
- "UltraLite NT_FILE connection parameter" on page 179
- "UltraLite CE_FILE connection parameter" on page 169

Opening UltraLite connections with connection strings

A connection string is a set of parameters that is passed from an application to the runtime so that a connection can be defined and established.

There are three steps that take place before a connection to a database is opened:

1. **The parameter definition phase**

   You must define the connection via a combination of supported parameters. Some connection parameters are always required to open a connection. Others are used to adjust database features for a single connection.

   How these parameters are supplied can vary depending on whether you are connecting from an UltraLite administration tool or an UltraLite application. See "Supplying UltraLite connection parameters" on page 35.

2. **The string assembly phase**

Either you or the application assembles the supplied parameters into a string. Connection strings contain a list of parameters defined as *keyword=value* pairs in a semicolon delimited list. See "Assembling parameters into UltraLite connection strings" on page 38.

For example, a connection string fragment that supplies a file name, user ID, and password is written as follows:

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

3. **The transmittal phase**

   When the connection string has been assembled, it is passed to the database via an UltraLite API to the UltraLite runtime for processing. If the connection attempt is validated, the connection is granted. Connection failures can occur if:

   - The database file does not exist.
   - Authentication was unsuccessful.

Assembling parameters into UltraLite connection strings

An assembly of connection parameters supplied in any application's connection code (be it an administration tool or a custom UltraLite application) is called a **connection string**. An application can parse the fields of a ConnectionParms object into a string, or you can type a connection string on a single line with the parameter names and values separated by semicolons:

```
parameter1=value1;parameter2=value2
```

The UltraLite runtime ensures that the parameters are assembled into a connection string before establishing a connection with it. 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 named database file until you supply this string:

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

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

Precedence of connection parameters for UltraLite administration tools

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

- O/S-specific options take precedence over nonspecific options. For example: CE_DBF takes precedence over DBF on CE devices
- If specified, the CE_FILE, desktop, device, and NT_FILE parameters always take precedence over DBF.
- 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.

Limitations

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 (;).

See also

- "Storing UltraLite parameters with the ULSQLCONNECT environment variable" on page 40

UltraLite user authentication

You cannot disable UltraLite user authentication. A successful connection requires that a user be authenticated. Unlike SQL Anywhere, UltraLite database users are created and managed solely for authentication and not for object ownership. Once a user authenticates and connects to the database, the user has unrestricted access to everything in that database, including schema data.

You can only add or modify UltraLite users from an existing connection. Therefore, any changes to your UltraLite user base can only occur after you have connected with a valid user ID and password.

If this is your first time connecting, the UID and PWD are the same values set when you first created the database. If you did not set an initial user, then you must authenticate with the defaults of **UID=DBA** and **PWD=sql**.

Bypass authentication

Although you cannot disable authentication, you can bypass it by using UltraLite defaults when you create and connect to the database.

If you do not supply the UID *and* the PWD parameters, UltraLite assumes the defaults of **UID=DBA** and **PWD=sql**.

To bypass authentication in UltraLite

1. Do not set the UID and PWD connection parameters when you create a database.
2. Do not delete or modify the default user in your UltraLite database.
3. Do not set the UID and PWD connection parameters when you connect to the database you have created.

See also

- "Limitations" on page 69
- "Working with UltraLite users" on page 69
- "Interpreting user ID and password combinations" on page 40

Interpreting user ID and password combinations

UltraLite allows you to set one, none, or both of the UID and PWD parameters—except when a partial definition prevents a user from being identified by UltraLite. The table below tells you how UltraLite interprets incomplete user definitions.

If you create a database with...	It has this impact...
No user ID *or* password.	UltraLite creates a default user with a UID of **DBA** and PWD of **sql**. You do not need to supply these connection parameters upon future connection attempts.
The user ID parameter only.	UltraLite creates a default user with a UID of **JaneD** and an empty PWD. When connecting, you must always supply the UID parameter. The PWD parameter is not required.
The password parameter only.	UltraLite generates an error. UltraLite cannot set a password without a user ID.

See also

- "UltraLite user authentication" on page 39
- "Limitations" on page 69
- "Working with UltraLite users" on page 69

Storing UltraLite parameters with the ULSQLCONNECT environment variable

The ULSQLCONNECT environment variable is optional, and is not set by the installation program. ULSQLCONNECT contains a list of parameters defined as *keyword=value* pairs in a semicolon delimited list.

Use ULSQLCONNECT to avoid having to supply the same connection parameters repeatedly to dbisql during development. You cannot use ULSQLCONNECT for custom applications.

Caution
Do not use the pound character (#) as an alternative to the equal sign; the pound character is ignored by dbisql. All platforms supported by UltraLite allow you to use = inside an environment variable setting.

To set ULSQLCONNECT for UltraLite desktop tools

1. Run the following command:

   set ULSQLCONNECT="*parameter=value*; ..."

2. If dbisql requires any additional parameters or if you need to override default values set with this environment variable, ensure you set these values. User supplied values always take precedence over this environment variable.

See also

- "Precedence of connection parameters for UltraLite administration tools" on page 38
- "Supplying UltraLite connection parameters" on page 35

Example

To use ULSQLCONNECT to connect to a file named *c:\database\myfile.udb* and authenticate the user **demo** with the password **test**, set the following variable in your ULSQLCONNECT environment variable:

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

By setting this environment variable, you no longer need to use the -c connection option for these defaults values—unless you need to override these values.

For example, if you were using ulload to add additional information to your database from an *extra.xml* file, you would run the following command:

```
ulload -a extra.xml
```

Deploying UltraLite to devices

In the majority of cases, development occurs on a Windows desktop 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.

Key considerations

There are two primary considerations for deploying your UltraLite solution:

- Deploying the files that provide UltraLite functionality (the runtime files)
- Deploying the UltraLite database file or files (used by the runtime files and containing application data)

Deploying the runtime files

Deploying the runtime files depends primarily on the language used to develop your solution (C/C++, .NET, M-Business, Objective C) and whether the UltraLite engine is required. The UltraLite engine is required if:

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

The alternative to the UltraLite engine is the UltraLite "in-process" runtime, which can be simpler to deploy and provide improved data access performance because process boundaries are not crossed (however it means that multiple applications cannot have connections open to the same database at the same time). The engine is available on:

- **Desktop** Windows (32 and 64 bit), Linux (32 bit) & Mac (64 bit)
- **Device** Windows Mobile devices

The engine executable can be found in *%sqlany%\ultralite\<platform>\<chip>\...*. In some cases, there is a version of the engine under *<chip>_dev* (for example *x86_dev*). This version of the engine contains development-time logging functionality that can be used to diagnose problems on platforms where it's difficult to debug. The logging output is intended to be used by Sybase engineers rather than customers. For production systems, use the version of the engine that is NOT in the *_dev* directory.

Deployment when the UltraLite engine is used

If your application uses the UltraLite engine, then you need to concern yourself with deploying all the files required by the engine, and all the files required by the client using it. The client files are programming-language specific. The engine files are independent of the programming language.

The engine consists of an executable (uleng12) and optional shared libraries that contain specific features. See "Deploy multiple UltraLite applications with the UltraLite engine" on page 44 and "UltraLite Engine utility (uleng12)" on page 194 for further information.

Deploying the UltraLite client files

The client files are closely associated with the application, and depend on the programming language of the application.

C++ development

When building a C/C++ application, you can either link all the UltraLite client code into your application or, if you are targeting Windows or Windows Mobile, you can load the client code dynamically at runtime. Linking the UltraLite client code into your application means you do not need to deploy any additional files with your application and the UltraLite engine. To link all the UltraLite client code into your application, link with:

Platform	Library	Additional files to deploy
Windows x64 and x86	*ulrtc.lib*	none

Platform	Library	Additional files to deploy
Windows Mobile/CE	*ulrtc.lib*	none
Linux	*libulrtc.a*	none
Not supported on Mac OS X/iPhone	N/A	N/A

If you want to load the UltraLite client code dynamically at runtime, link your application code with:

Platform	Library	Additional files to deploy
Windows x64 and x86	*ulimpc.lib*	*ulrtc12.dll*
Windows Mobile/CE	*ulimpc.lib*	*ulrtc12.dll*
Not supported on Linux	N/A	N/A
Not supported on Mac OS X/iPhone	N/A	N/A

.NET development

When building a .NET application using the UltraLite client, be sure to deploy *ulnetclient12.dll* along with your .NET assemblies. See "System requirements and supported platforms" [*UltraLite - .NET Programming*] for more information on required files for deployment.

Deployment when the in-process version of UltraLite is used

When building a C/C++ application that does not use the UltraLite engine, you can either link to a static UltraLite runtime library (this makes sure all the UltraLite code is linked into your application) or, on Windows and Windows Mobile, you can link to an import library and load the UltraLite runtime code dynamically when the application starts. For Windows and Windows Mobile, if you link to the import library, then you need to deploy *ulrt12.dll* along with your application. For more information on compiling and linking your application, see "Developing applications using the UltraLite C++ API" [*UltraLite - C and C++ Programming*]. If you are linking with the static runtime library, there are no additional files to deploy with your application.

Initial installation

Initially installing your UltraLite solution on a device is a required step so that the device can be continuously maintained.

See also

- "Using MobiLink file transfers" on page 106
- "UltraLite clients" on page 93
- "Using ActiveSync with UltraLite on Windows Mobile" on page 108
- "Deploying UltraLite schema upgrades" on page 51
- "ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]" on page 368

Deploying UltraLite databases

Once you get the UltraLite binaries deployed properly, you will want to determine how to get an initial UltraLite database file on the device. There are several ways to get your initial UltraLite database file:

- Central Administration can be used to send down the initial UDB file, or send down a command to create the initial database file and execute SQL to give it schema. See "Central administration of remote databases" [*MobiLink - Server Administration*].
- The application can be programmed to create the initial database file.
- The method to deploy the application can include the initial database file in its file list.
- The application can be programmed to download the initial file if it doesn't exist.

Deploying upgrades to the UltraLite runtime files

When you change your application to use an upgraded version of the UltraLite runtime files (for example, when you move from UltraLite version 11 to version 12) you will most likely need to rebuild your application and re-link with the required UltraLite runtime libraries or assemblies. When you link with a new version of the UltraLite libraries, you should be sure to deploy all updated versions of the UltraLite files as described above. Note also that UltraLite clients must closely match with UltraLite engines (the major versions must match). See "Upgrading databases created with previous versions of UltraLite" [*SQL Anywhere 12 - Changes and Upgrading*].

Deploying changes to the UltraLite database files

When your application changes and you need to change the schema of your database, you can use Central Administration to send a new version of the UltraLite database file to the device running the application. See "Central administration of remote databases" [*MobiLink - Server Administration*].

Alternatively, you can write logic in your application that performs the upgrade.

Deploy multiple UltraLite applications with the UltraLite engine

The UltraLite engine is a data management module that manages concurrent UltraLite database connections from applications on Windows (32 and 64 bit), Linux (32 bit) & Mac (64 bit). The engine is automatically installed to the desktop with the SQL Anywhere installer. Therefore, you only need to deploy the engine to Windows Mobile devices, if required.

To deploy uleng to a Windows Mobile device

1. Copy the *uleng12.exe* file and the appropriate **.dll* files. The **.dll* files you copy should include any database encryption, synchronization encryption, or compression **.dll* files required.

Filename	Basic	ECC TLS	RSA TLS	FIPS RSA TLS	HTTPS	Compression	FIPS AES data encryption
uleng12.exe	X	X	X	X	X	X	X
mlcecc12.dll [1]		X			X		
mlcrsa12.dll[1]			X		X		
mlcrsafips12.dll				X	X		
mlczlib12.dll						X	
sbgse2.dll				X	X		X
ulfips12.dll							X
ulrt12.dll[2]	X	X	X	X	X	X	X

[1] File not needed if an application links directly against *ulecc.lib* and *ulrsa.lib* respectively.

[2] File only needed if an application links against *ulimp.lib*.

2. Save the files to an appropriate directory. Typically you use one of the following destination directories:
   - The *\windows* directory. This location is the recommended location, as the client automatically looks for the engine in this location. See "Starting the UltraLite engine" on page 45. for a complete list of where UltraLite looks for the engine.
   - The directory for other UltraLite application files.
3. If you use any location other than the *\windows* directory, include the START connection parameter. This parameter starts the UltraLite engine when the application connects the UltraLite database.

   For example, a connection string to the database or connection code for a Windows Mobile client application, might use this START parameter:

   ```
   "START=\Program Files\MyApp\uleng12.exe"
   ```

See also

- "UltraLite START connection parameter" on page 182
- "Deploy UltraLite with AES_FIPS database encryption" on page 46
- "Deploy UltraLite with TLS-enabled synchronization" on page 47

Starting the UltraLite engine

An UltraLite client application can automatically start the UltraLite engine, and will need to do that unless it explicitly provides the location of the engine (see "UltraLite START connection parameter" on page 182).

When attempting to automatically start the UltraLite engine, an UltraLite client will look in the following locations (depending on the platform):

Technique	Windows Mobile
Windows desktop	1. Directory of the application that's starting it 2. The current working directory 3. The system path 4. The SQL Anywhere install directory (either under *bin32* or *bin64*), depending on whether the client is 32-bit or 64-bit
Windows Mobile/CE	1. *Windows*\ 2. \ (the root directory) 3. *UltraLiteDB*\
Linux	1. The directory of the application that's auto-starting it 2. *install-dir/bin32*
Mac	N/A
iPhone	N/A

Deploy UltraLite with AES_FIPS database encryption

Strong database encryption technology makes a database inoperable and inaccessible without a key (a type of a password). An algorithm encodes the information contained in your database and transaction log files so they cannot be deciphered. However, database encryption requires that you deploy the appropriate number of files with your database.

When you connect to UltraLite with the -fips option, you can run databases encrypted with AES or AES_FIPS strong encryption. To ensure you are running with AES_FIPS, use **-fips=1**.

If you are encrypting your database with AES FIPS encryption, you must configure and deploy your device for each platform.

To set up your application and device for an AES FIPS encrypted UltraLite database

1. Create an UltraLite database with the property **fips=1**. See "UltraLite fips creation parameter" on page 142.
2. Use the following connection parameter in your application's connection string: **DBKEY=*key***. See "UltraLite DBKEY connection parameter" on page 174.

3. Ensure that you deploy the appropriate files to your device.

   Windows desktop, Windows Mobile, require *ulfips12.dll* and *sbgse2.dll*. The Windows Mobile component also requires the component DLL files.

See also

- "Deploy UltraLite with TLS-enabled synchronization" on page 47
- UltraLite.NET: "Encryption and obfuscation" [*UltraLite - .NET Programming*]
- UltraLite for C++: "Encrypting data" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Database encryption and obfuscation" [*UltraLite - M-Business Anywhere Programming*]
- "UltraLite fips creation parameter" on page 142
- "Securing UltraLite databases" on page 32

Deploy UltraLite with TLS-enabled synchronization

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

UltraLite client applications of MobiLink must be configured to use enable TLS synchronization. Transport-layer security enables encryption, tamper detection, and certificate-based authentication. See "Transport-layer security" [*SQL Anywhere Server - Database Administration*].

Platform support
RSA, ECC, and FIPS encryption are not available on all platforms. For information about which platforms support which encryption method, see http://www.sybase.com/detail?id=1061806.

To set up TLS synchronization on an UltraLite client application and device

1. Enable encrypted synchronization by calling one of the following in your application code:
   - To enable RSA encryption, call ULEnableRsaSyncEncryption. See "ULEnableRsaSyncEncryption method" [*UltraLite - C and C++ Programming*].
   - To enable ECC encryption, call ULEnableEccSyncEncryption. See "ULEnableEccSyncEncryption method" [*UltraLite - C and C++ Programming*].
2. Set the synchronization information stream to either TLS or HTTPS.
3. If you are enabling ECC or FIPS encryption, you also need to:
   - **ECC** Set the **tls_type** network protocol option to **ECC**. See "tls_type" [*MobiLink - Client Administration*].
   - **FIPS** Set the **fips** network protocol option to **Yes**. See "fips" [*MobiLink - Client Administration*].

4. Ensure that you have linked to the appropriate libraries:

Platform	Linking	RSA encryption	ECC encryption	FIPS encryption
Windows desktop	static[1]	*ulrsa.lib*	*ulecc.lib*	none
Windows desktop	dynamic[2]	none	none	none
Windows Mobile	static[1]	*ulrsa.lib*	*ulecc.lib*	none
Windows Mobile	dynamic [1]	none	none	none

[1] You must also link to *ulimp.lib*.

5. Ensure that the appropriate files are copied to the device:

Platform	Linking	RSA encryption	ECC encryption	FIPS encryption
Windows desktop	static	none	none	*mlcrsafips12.dll* *sbgse2.dll*
Windows desktop	dynamic[1]	*mlcrsa12.dll*	*mlcecc12.dll*	*mlcrsafips12.dll* *sbgse2.dll*
Windows Mobile	static	none	none	*mlcrsafips12.dll* *sbgse2.dll*
Windows Mobile	dynamic[1]	*mlcrsa12.dll*	*mlcecc12.dll*	*mlcrsafips12.dll* *sbgse2.dll*
Windows Mobile components and UltraLite engine	static[2]	*mlcrsa12.dll*	*mlcecc12.dll*	*mlcrsafips12.dll* *sbgse2.dll*

[1] You must also deploy *ulrt12.dll*.

[2] You must also deploy your component *.dll* file and/or *uleng12.exe*.

See also

- "Configuring UltraLite clients to use transport-layer security" [*SQL Anywhere Server - Database Administration*]
- UltraLite.NET: "Encryption and obfuscation" [*UltraLite - .NET Programming*]
- UltraLite for C++: "Encrypting data" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Database encryption and obfuscation" [*UltraLite - M-Business Anywhere Programming*]

Deploy the ActiveSync provider for UltraLite

The UltraLite ActiveSync provider is a software module that let users gain access to their devices from the desktop. Like other software components, you need to deploy the necessary files to the device to ensure that UltraLite operates with Windows Mobile ActiveSync.

During development you install UltraLite onto your desktop with the SQL Anywhere installer. However, when you deploy UltraLite to the end user, you must manually install and register the ActiveSync provider on the end user's computer. This requirement ensures that ActiveSync knows when to call a specific instance of a provider for a specific application.

- **mlasinst.exe** Installs the ActiveSync provider and registers it with the ActiveSync Manager. This utility also registers applications with the ActiveSync provider for synchronization.
- **mlasdesk.dll** The DLL that is loaded by the ActiveSync Manager on the desktop. *mlasinst.exe* registers the location of this file with the ActiveSync Manager.
- **mlasdev.dll** The DLL that is loaded by the ActiveSync Manager on the device. *mlasinst.exe* deploys this file to the correct location on the device.
- **dblgen12.dll** The language resource library.

For a list of supported provider platforms, see http://www.sybase.com/detail?id=1002288.

To install ActiveSync applications

1. Ensure that the end-user has:
   - The ActiveSync Manager installed.
   - The ActiveSync provider files copied from a development computer to the user's hard drive.
2. Run mlasinst to install a provider for ActiveSync. You can also use it to register and deploy the UltraLite application to the user's Windows Mobile device—depending on the command line syntax you use. If your UltraLite application uses multiple files, you must manually copy the required files.

   The following example assumes that both *mlasdesk.dll* and *mlasdev.dll* are in the current directory. The -k and -v options are used. The -p and -x options are command line options for the application when it is started by ActiveSync.

   ```
   mlasinst "C:\My Files\myULapp.exe" "\Program Files\myULapp.exe"
      "My Application" MYAPP -p -x -v -k
   ```

If you were to use this utility to deploy a pre-compiled CustDB for the ARM 5.0 processor, the command line would be similar to the following one:

```
mlasinst -v "install-dir\UltraLite\ce\arm.50"
"install-dir\UltraLite\ce\arm.50\custdb.exe" custdb.exe CustDB CUSTDBDEMO
```

Note
You can also use the ActiveSync to register your UltraLite application at a later time if you choose. See "Register applications with the ActiveSync Manager" on page 50.

3. Restart your computer so ActiveSync can recognize the new provider.
4. Enable the MobiLink provider.
   a. From the ActiveSync window, click **Options**.
   b. Check **MobiLink Clients** in the list and click **OK** to activate the provider.
   c. To see a list of registered applications, click **Options**, choose **MobiLink Clients**, and click **Settings**.

See also

- "Register applications with the ActiveSync Manager" on page 50
- "Microsoft ActiveSync Provider Installation utility (mlasinst)" [*MobiLink - Client Administration*]

Register applications with the ActiveSync Manager

You can register your application for use with ActiveSync either by using the ActiveSync Provider Installation utility or using the ActiveSync Manager itself. This section describes how to use the ActiveSync Manager.

To register an application for use with the ActiveSync Manager

1. Launch ActiveSync.
2. From the ActiveSync window, choose **Options**.
3. From the list of information types, choose **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 ActiveSync user interface.
   - **Class name** The registered class name for the application. See "Assigning class names for applications" [*UltraLite - C and C++ Programming*].
   - **Path** The location of the application on the device.

- **Arguments** Any command line arguments to be used when ActiveSync starts the application.

6. Click **OK** to register the application.

See also

- "Microsoft ActiveSync Provider Installation utility (mlasinst)" [*MobiLink - Client Administration*]

Deploying UltraLite schema upgrades

To make schema upgrades, use the SQL statement ALTER DATABASE SCHEMA FROM FILE.

The upgrade process

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."

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 will be dropped. For example, if you have a VARCHAR column and convert it to an INT column and the value for a row is **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.

To upgrade the UltraLite schema

1. Create a SQL script of DDL statements that define the new schema. The character set of the SQL script file must match the character set of the database you want to upgrade.

   You should use either ulinit or ulunload 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.

   - If you are using ulunload, use the -n and -s [*file*] options.
   - If you are using ulinit, use the -l [*file*] option.

If you do not use either ulunload or ulinit, review the script and ensure that:

- You do not rename tables, columns, or publications. RENAME operations are not supported. If you rename a table, it is processed as a DROP TABLE and CREATE TABLE operation.
- You have not included non-DDL statements. Including non-DDL statements may not have the effect you expect.
- Words in the SQL statement are separated by spaces.
- Only one SQL statement can appear in each line.
- Comments are prepended with double hyphens (-), and only occur at the start of a line.

2. Backup the database against which the upgrade will be performed.
3. Run the new statement. For example:

```
ALTER DATABASE SCHEMA FROM FILE 'MySchema.sql';
```

Error notification

Because UltraLite error callback is active during the upgrade process, you are notified of errors during the conversion process. For example, SQLE_CONVERSION_ERROR reports all values that could not be converted in its parameters. Errors do *not* mean the process failed. The final SQL code after the statement returns is a 130 warning in this case. These warnings describe operations of the conversion process and do not stop the upgrade process.

See also

- “ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]” on page 368
- “UltraLite Initialize Database utility (ulinit)” on page 197
- “UltraLite Database Unload utility (ulunload)” on page 214
- “UltraLite SQL statements” on page 366
- “Comments in UltraLite” on page 226

Working with UltraLite databases

Working with 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 Sybase 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.

Row packing and table definitions

UltraLite works with rows in two formats:

- **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 entirely 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.

Note
You cannot declare tables to be any large size you require. 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 SQLE_MAX_ROW_SIZE_EXCEEDED (-1132).

See also

- "UltraLite page_size creation parameter" on page 146
- "Database tables" [*SQL Anywhere 12 - Introduction*]
- "Creating a SQL Anywhere database" [*SQL Anywhere Server - Database Administration*]
- "UltraLite system tables" on page 219

Creating UltraLite tables

You can create new tables to hold your relational data, either with SQL statements in Interactive SQL or with Sybase Central.

In UltraLite, you can only create base tables, which you declare to hold persistent data. The table and its data continue to exist until you explicitly delete the data or drop the table. UltraLite does not support global temporary or declared temporary tables.

Note
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.

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected database.

To create an UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. In the left pane, right-click **Tables** and choose **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, choose **Save Table**.

Interactive SQL

In Interactive SQL, you can declare columns while creating a new table.

To create an UltraLite table (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a CREATE TABLE statement.

   For example, the following statement creates a new table to describe the various skills and professional aptitudes of employees within a company. The table has columns to hold an identifying number, a name, and a type (for example, technical, or administrative) for each skill.

```
CREATE TABLE Skills (
   SkillID INTEGER PRIMARY KEY,
   SkillName CHAR( 20 ) NOT NULL,
   SkillType CHAR( 20 ) NOT NULL
);
```

See also

- “CREATE TABLE statement [UltraLite] [UltraLiteJ]” on page 384
- “Adding a column to an UltraLite table” on page 55

Using allsync and nosync suffixes

You can append either **_allsync** or **_nosync** to a table name to control data restriction for synchronization. You can use these suffixes as an alternative to using publications to control data restrictions. To control data priority, define one or more publications.

- If you create a table with a name ending in **_allsync**, all rows of that table are synchronized at each synchronization—even if they have not changed since the last synchronization.

Tip
You can store user specific or client specific data in allsync tables. You can then upload the data in the UltraLite table to a temporary table in the consolidated database on synchronization. Synchronization scripts can control the data and save you from having to maintain that data in the consolidated database.

- If you create a table with a name ending in **_nosync**, all rows of that table are excluded from synchronization. You can use these tables for persistent data that is not required in the consolidated database's table.

See also

- "Working with UltraLite publications" on page 65
- "Designing synchronization in UltraLite" on page 99
- "Non-synchronizing tables in UltraLite" on page 101
- "Allsync tables in UltraLite" on page 102
- "UltraLite CustDB samples" on page 72

Example

In the *CustDB.udb* sample database, you can see that one table was declared a nosync table because the table name is named ULIdentifyEmployee_nosync. Therefore, no matter how data changes in this table, it is never synchronized with MobiLink and information will not appear in the *CustDB.db* consolidated database.

Adding a column to an UltraLite table

You can add a new column easily if the table is empty. However, if the table already holds data, you can only add a column if the column definition includes a default value or allows NULL values.

You can use either Sybase Central or execute a SQL statement (for example, Interactive SQL) to perform this task.

Sybase Central

In Sybase Central, you can perform this task while working with a selected table.

To add a new column to an UltraLite table (Sybase Central)

1. 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 choose **New** » **Column**.
5. Set the attributes for the new column.
6. From the **File** menu, choose **Save Table**.

Interactive SQL

In Interactive SQL, you can only declare columns while creating or altering a table.

To add columns to a new UltraLite table (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a CREATE TABLE statement or ALTER TABLE, and define the columns by declaring the name, and other attributes.

   The following example 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)
      );
   ```

   The following example modifies the customer table to now include a column for addresses that can hold up to 50 characters:

   ```
   ALTER TABLE customer
   ADD address CHAR(50);
   ```

See also

- "Choosing object names" [*SQL Anywhere Server - Database Administration*]
- "Data types in UltraLite" on page 230
- "Choosing column data types" [*SQL Anywhere Server - Database Administration*]
- "CREATE TABLE statement [UltraLite] [UltraLiteJ]" on page 384
- "ALTER TABLE statement [UltraLite] [UltraLiteJ]" on page 371

Altering UltraLite column definitions

You can change the structure of column definitions for a table by altering various column attributes, or even deleting columns entirely. 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.

You can use either Sybase Central or Interactive SQL to perform this task.

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected table.

To alter an existing UltraLite column (Sybase Central)

1. 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, choose **Save Table**.

Interactive SQL

In Interactive SQL, you can perform these tasks with the ALTER TABLE statement.

To alter an existing UltraLite column (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute an ALTER TABLE statement.

The following examples show how to change the structure of the database. In all these cases, the statement is committed immediately. So, once you make the change, any item referring to this table may no longer work.

The following statement shortens the SkillDescription column from a maximum of 254 characters to a maximum of 80:

```
ALTER TABLE Skills
MODIFY SkillDescription CHAR( 80 );
```

The following statement deletes the Classification column:

```
ALTER TABLE Skills
DROP Classification;
```

The following statement changes the name of the entire table:

```
ALTER TABLE Skills
RENAME Qualification;
```

See also

- "Choosing object names" [*SQL Anywhere Server - Database Administration*]
- "Data types in UltraLite" on page 230
- "Choosing column data types" [*SQL Anywhere Server - Database Administration*]
- "ALTER TABLE statement [UltraLite] [UltraLiteJ]" on page 371

Deleting UltraLite tables

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.

You can use either Sybase Central or Interactive SQL to perform this task.

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected table.

To delete an UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. In the left pane, double-click **Tables**.
3. Right-click a table and choose **Delete**.
4. Click **Yes**.

Interactive SQL

In Interactive SQL, deleting a table is also called dropping it. You can drop a table by executing a DROP TABLE statement.

To delete an UltraLite table (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a DROP TABLE statement.

   For example, the following DROP TABLE statement deletes all the records in the Skills table and then removes the definition of the Skills table from the database:

   ```
   DROP TABLE Skills;
   ```

   Like the CREATE statement, the DROP statement automatically executes a COMMIT statement before and after dropping the table. This statement makes all changes to the database since the last COMMIT or ROLLBACK permanent. The DROP statement also drops all indexes on the table.

See also

- "DROP TABLE statement [UltraLite] [UltraLiteJ]" on page 394

Browsing the information in UltraLite tables

You can use Sybase Central or Interactive SQL to browse the data held within the tables of an UltraLite database. 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.

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected database.

To browse UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. If system tables are hidden and you want to browse the data in one or more tables, right-click the white space of the **Contents** pane and choose **Show System Objects**.
3. To view a list of tables, double-click **Tables**.
4. To view table data, double-click a table and click the **Data** tab in the right pane.

To filter UltraLite system tables (Sybase Central)

1. Connect to the UltraLite database.
2. Right-click the database you are connected to and choose either **Hide System Objects** or **Show System Objects**.

Interactive SQL

In Interactive SQL, you can perform these tasks with the SELECT statement.

To browse UltraLite user tables (Interactive SQL)

1. Connect to a database.
2. Execute a SELECT statement, specifying the user table you want to browse.

To browse UltraLite system tables (Interactive SQL)

1. Connect to a database.
2. Execute a SELECT statement, by the system table you want to browse.

   For example, to display the contents of the table systable on the **Results** tab in the **Results** pane in Interactive SQL, execute the following command:

   ```
   SELECT * FROM SYSTABLE;
   ```

See also

- "UltraLite system tables" on page 219

Copying and pasting data to or from UltraLite databases

With Sybase 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 as 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.

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.

Sybase 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 Sybase 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.

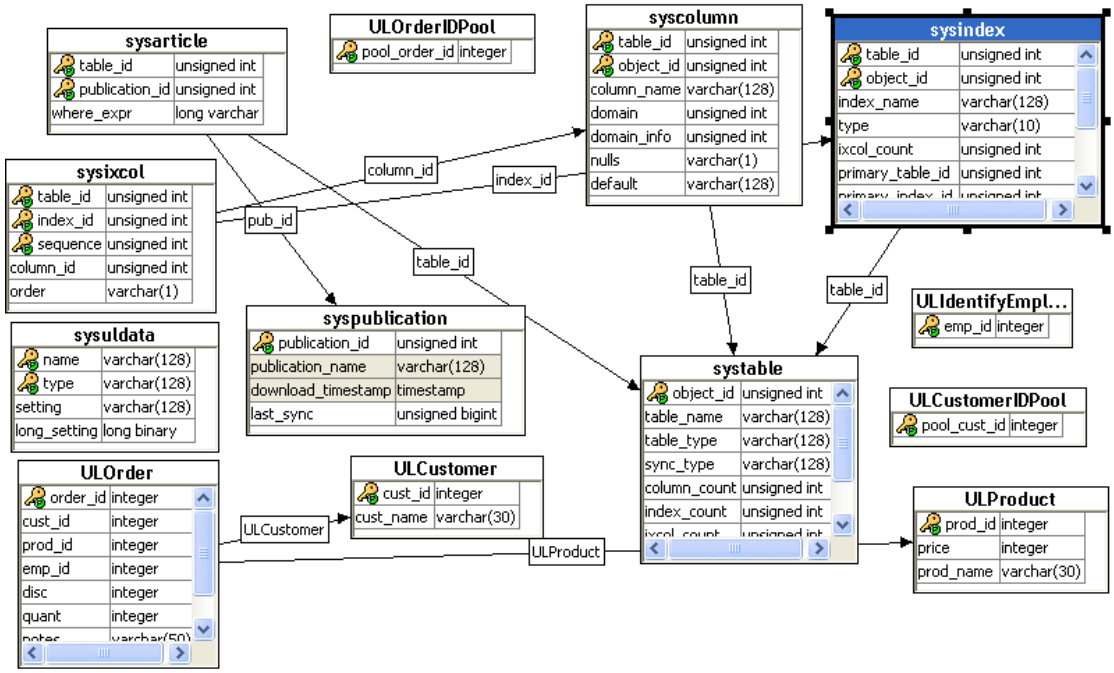
- Use the SELECT statement results into a named object.
- Use the INSERT statement to insert a row or selection of rows from elsewhere in the database into a table.

See also

- “Copying database objects in the SQL Anywhere 12 plug-in” [*SQL Anywhere Server - Database Administration*]
- “INSERT statement [UltraLite] [UltraLiteJ]” on page 397
- “SELECT statement [UltraLite] [UltraLiteJ]” on page 402

Viewing entity-relationship diagrams from the UltraLite plug-in

When you are connected to a database from the UltraLite plug-in, you can view an entity-relationship diagram of the tables in the database. When you have the database selected, click the **ER Diagram** tab in the right pane to see the diagram.



When you rearrange objects in the diagram, the changes persist between Sybase Central sessions. Double-clicking a table takes you to the column definitions for that table.

See also

- "Creating a SQL Anywhere database" [*SQL Anywhere Server - Database Administration*]

Working with UltraLite indexes

An index provides an ordering (either ascending or descending) of a table's rows based on the values in one or more columns. 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.

Performance tip
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.

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.

See also

- "Using index scans" on page 82
- "Execution plans in UltraLite" on page 262
- "Composite indexes" [*SQL Anywhere Server - SQL Usage*]
- UltraLite.NET: "Accessing and manipulating data with the Table API" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Prepare method" [*UltraLite - .NET Programming*]
- UltraLite for C++: "Accessing data using ULTable class" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "ULPreparedStatement class" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Working with data using the Table API" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "PreparedStatement class" [*UltraLite - M-Business Anywhere Programming*]

When to use an index

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 as 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.

Choosing an index type

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.

Index characteristic	Unique keys	Unique indexes	Non-unique indexes
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

Notes

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.

See also

- "Adding UltraLite indexes" on page 64

Adding UltraLite indexes

You can use either Sybase Central or Interactive SQL to perform this task.

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

Sybase Central

In Sybase Central, you can perform this task while working with a selected database.

To create a new index for a given UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. Right-click **Indexes**, and choose **New** » **Index**.
3. Follow the instructions in the wizard.

Interactive SQL

In Interactive SQL, you can perform this task with the CREATE INDEX statement.

To create a new index for a given UltraLite table (Interactive SQL)

1. Connect to an UltraLite database.
2. Execute a CREATE INDEX statement.

   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. See "CREATE INDEX statement [UltraLite] [UltraLiteJ]" on page 376.

   For 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;
```

See also

- "CREATE INDEX statement [UltraLite] [UltraLiteJ]" on page 376

Dropping an index

Dropping an index deletes it from the database.

You can use either Sybase Central or Interactive SQL to perform this task.

Sybase Central

In Sybase Central, you can perform this task while working with a selected database.

To drop an UltraLite index (Sybase Central)

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

Interactive SQL

In Interactive SQL, deleting an index is also called dropping it. You can perform this task with the DROP INDEX statement.

To drop an UltraLite index (Interactive SQL)

1. Connect to a database.
2. Execute a DROP INDEX statement.

   For example, the following statement removes the EmployeeNames index from the database:

   ```
   DROP INDEX EmployeeNames;
   ```

See also

- "DROP INDEX statement [UltraLite] [UltraLiteJ]" on page 392

Working with UltraLite publications

A publication is a database object that identifies the data that is to be synchronized. If you want 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 synchronization logic you require. 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 Sybase Central or with the CREATE PUBLICATION statement. In Sybase 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.

See also

- "Table order in UltraLite" on page 103
- "Publishing data" [*MobiLink - Client Administration*]
- "Designing synchronization in UltraLite" on page 99
- "Writing synchronization scripts" [*MobiLink - Server Administration*]

Publishing whole tables in UltraLite

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 Sybase Central or Interactive SQL to perform this task.

Sybase Central

In Sybase Central, you can perform this task while working with the connected database.

To publish one or more whole UltraLite tables (Sybase Central)

1. Connect to the UltraLite database.
2. Right-click the **Publications** folder, and choose **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, select tables from the **Available Tables** list. Click **Add**.
5. Click **Finish**.

Interactive SQL

In Interactive SQL, you can perform this task with the CREATE PUBLICATION statement.

To publish one or more whole UltraLite tables (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a CREATE PUBLICATION statement that specifies the name of the new publication and the table you want to publish.

   For example, the following statement creates a publication that publishes the whole customer table:

   ```
   CREATE PUBLICATION pub_customer (
      TABLE customer
   );
   ```

See also

- "CREATE PUBLICATION statement [UltraLite] [UltraLiteJ]" on page 378
- "UltraLite clients" on page 93

Publishing a subset of rows from an UltraLite table

A publication can only contain specific table rows. In Sybase Central or Interactive SQL, 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.

Sybase Central

In Sybase Central, you can perform this task while working with the connected database.

To publish only some rows in an UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. Right-click the **Publications** folder, and choose **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, select a table and click **Add**.
6. Click the **WHERE Clauses** tab, and select the table from the **Articles** list. Optionally, you can use the Insert window to assist you in formatting the search condition.
7. Click **Finish**.

Interactive SQL

In Interactive SQL, you can perform this task with the CREATE PUBLICATION statement.

To create a publication in UltraLite using a WHERE clause (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a CREATE PUBLICATION statement that includes the tables you want to include in the publication and a WHERE condition.

   For example, the following example creates a single-article publication that includes all sales order information for sales rep number 856:

```
CREATE PUBLICATION pub_orders_samuel_singer
 ( TABLE SalesOrders
     WHERE SalesRepresentative = 856 );
```

See also

- "CREATE PUBLICATION statement [UltraLite] [UltraLiteJ]" on page 378
- "UltraLite clients" on page 93

Dropping a publication for UltraLite

You can drop a publication using either Sybase Central or Interactive SQL.

Sybase Central

In Sybase Central, you can perform this task while working with the connected database.

To drop a publication (Sybase Central)

1. Connect to the UltraLite database.
2. In the left pane, double-click the **Publications** folder.
3. Right-click the publication and choose **Delete**.
4. Click **Yes**.

Interactive SQL

In Interactive SQL, deleting a publication is also called dropping it. You can perform this task with the DROP PUBLICATION statement.

To drop a publication (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a DROP PUBLICATION statement.

   For example, the following statement drops the publication named pub_orders:


```
DROP PUBLICATION pub_orders;
```

See also

- "DROP PUBLICATION statement [UltraLite] [UltraLiteJ]" on page 392
- "UltraLite clients" on page 93

Working with UltraLite users

User IDs and passwords are encrypted in the UltraLite database, so you can only view the list of defined users in Sybase Central.

UltraLite user IDs are not the same as MobiLink user names or SQL Anywhere user IDs.

Limitations

When creating unique user IDs, bear the following limitations in mind:

- UltraLite supports up to four unique users per database.
- Both the user ID and password values have a limit of 31 characters.
- Passwords are always case sensitive and user IDs are always case insensitive. You can change a password anytime from Sybase Central.
- Any leading or trailing spaces the user ID are ignored. The user ID cannot include leading single quotes('), leading double quotes ("), or semicolons(;).
- You cannot change a user ID once it is created. Instead, you must delete the user ID and then add a new one.
- Passwords can be changed using Sybase Central.

Adding a new UltraLite user

UltraLite does not support the creation of users with Interactive SQL. However, you can add users by:

- Using Sybase Central to add users to the User folder.
- Using the GrantConnectTo function on the Connection object to add new users from an UltraLite application.

To create a new UltraLite user (Sybase Central)

1. Connect to the UltraLite database.
2. Right-click the **Users** folder, and choose **New** » **User**.

3. Follow the instructions in the wizard. Ensure you understand how UltraLite interprets different user ID and password combinations. See "Interpreting user ID and password combinations" on page 40.

See also

- UltraLite.NET: "GrantConnectTo method" [*UltraLite - .NET Programming*]
- UltraLite C/C++ : "GrantConnectTo method" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "grantConnectTo method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for embedded SQL: "ULGrantConnectTo method" [*UltraLite - C and C++ Programming*]

Deleting an existing UltraLite user

UltraLite does not support the deletion of users using a SQL statement. However, you can delete users by using:

- Sybase Central to delete users from the User folder.
- The RevokeConnectFrom function on the Connection object to remove users from an UltraLite application.

To delete an existing UltraLite user (Sybase Central)

1. Connect to the UltraLite database.
2. In the left pane, double-click the **Users** folder.
3. Right-click the user and choose **Delete**.

See also

- UltraLite.NET: "RevokeConnectFrom method" [*UltraLite - .NET Programming*]
- UltraLite C/C++ : "RevokeConnectFrom method" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "revokeConnectFrom method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for embedded SQL: "ULRevokeConnectFrom method" [*UltraLite - C and C++ Programming*]

Working with event notifications

UltraLite now supports events and notifications. A notification is a message that is sent when an event occurs, also providing additional parameter information. UltraLite has system events and events can also be user-defined.

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

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
Commit	Signaled upon completion of a commit.
SyncComplete	Signaled upon completion of a sync.
TableModified	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, SQLE_EVENT_NOTIFICATION_QUEUE_FULL is signaled and the pending notification is discarded.

UltraLite CustDB samples

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

CustDB 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.

Different versions of the application code exist for each supported programming interface and platform. However, the tutorial references the compiled version of the application for Windows desktops only. Remember that each version varies to conform to the conventions of each platform.

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.

After following each lesson you will know how to:

- Run the MobiLink server to carry out data synchronization between the consolidated database and the UltraLite remote.
- Use Sybase Central to browse the data in the UltraLite remote.
- Manage UltraLite databases with UltraLite command line utilities.

See also

- "CustDB sample overview" on page 73
- "CustDB Scenario" [*MobiLink - Getting Started*]
- "Users in the CustDB sample" [*MobiLink - Getting Started*]
- "Tables in the CustDB databases" [*MobiLink - Getting Started*]

CustDB sample overview

SQL Anywhere CustDB database

This is the consolidated database. During installation, an ODBC data source called SQL Anywhere 12 CustDB is created for this database. The database file is located at *samples-dir\UltraLite\CustDB\makedbs.cmd* (*makedbs.sh* for Mac or Linux).

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: *samples-dir\UltraLite\CustDB\newdb.bat*

For more information about the schema of this file, see "Exploring the CustDB sample for MobiLink" [*MobiLink - Getting Started*].

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

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: *samples-dir\UltraLite.NET\CustDB\Common*
- For all other platforms and APIs: *samples-dir\UltraLite\CustDB\custdb.udb*

The UltraLite database is also recreated by the *makedbs.cmd* script (*makedbs.sh* for Mac or Linux).

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

RDBMS-specific build scripts

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

In the *samples-dir\MobiLink\CustDB* directory, you can find the following files:

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

For more information about setting up a consolidated database, see "Setting up the CustDB consolidated database" [*MobiLink - Getting Started*].

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

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. Although the contents of these files vary, they all include an outline the steps required to build and run the sample.

The location the applications and its ReadMe depends on your development environment. See "Lesson 1: Build and run the CustDB application" on page 75.

Synchronization logic

The UltraLite database SQL statements and synchronization calls are located in *custdbcpp.cpp* for the C++ API and in *custdb.sql* for ESQL.

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

See also

- "MobiLink consolidated databases" [*MobiLink - Server Administration*]

Lesson 1: Build and run the CustDB application

The CustDB application is built for many development environments. For a general procedure that applies to all environments, see the following section.

To build and run the CustDB application

1. Build the CustDB application:
   a. Open a CustDB project file in the appropriate environment.
   b. Compile the source code.
2. Run the CustDB application:
   a. Deploy the CustDB executable file to the mobile device.
   b. Deploy the UltraLite CustDB database to the mobile device.
   c. Run the CustDB executable file.

UltraLite for 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 *install-dir\UltraLite\Windows\x86* directory.

UltraLite for C/C++

- **All versions of C/C++** You can find multiple versions of the C/C++ CustDB project file because of the many C/C++ development environments. Most versions make use of the generic files. These files are located in the *samples-dir\UltraLite\Custdb* directory.

For information about all versions of C/C++ CustDB applications, see *samples-dir\UltraLite\Custdb \readme.txt*.

- **Visual Studio** You can find project files in the *samples-dir\UltraLite\CustDB\vs9* and *samples-dir \UltraLite\CustDB\vs8* directories depending on your version of Visual Studio. To build and run the CustDB application, follow the instructions given in the beginning of the lesson.

- **Xcode for iPhone** For development on Mac OS X and iPhone, see *samples/ultralite/custdb/iphone*.

UltraLite.NET

You can find project files specific to Microsoft Visual Studio in the *samples-dir\UltraLite.NET\CustDB* directory.

For instructions on building the CustDB application for Windows Mobile using Microsoft Visual Studio*samples-dir\UltraLite.NET\CustDB\CE\ReadMe.html*.

To obtain deployment directory information for Microsoft Windows desktop, and information on where to download additional UltraLite.NET samples, see *samples-dir\UltraLite.NET\CustDB\Desktop \ReadMe.html*.

UltraLite for M-Business Anywhere

You can find project files specific to M-Business Anywhere in the *samples-dir \UltraLiteForMBusinessAnywhere\CustDB* directory.

For more information about building the CustDB application using M-Business Anywhere, see "UltraLite for M-Business Anywhere quick start" [*UltraLite - M-Business Anywhere Programming*]. The instructions are applicable to Windows Mobile and Windows.

Lesson 2: Log in and populate the UltraLite remote database

This lesson demonstrates how to:

- Start the sample MobiLink server.
- Start the sample UltraLite client application.
- Log into UltraLite.

In this tutorial, the sample application is running on the same desktop computer as the MobiLink server. However, you can deploy a client application to the device and achieve the same result.

To start and synchronize the sample application

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **MobiLink** » **Synchronization Server Sample**. Or, execute the following command:

   ```
   mlsrv12 -c "DSN=SQL Anywhere 12 CustDB" -vcrs
   ```


(Use *mobilink.sh* on Mac OS X or Linux.)

The window displays messages about the MobiLink server's status.

2. Choose **Start** » **Programs** » **SQL Anywhere 12** » **UltraLite** » **Windows Sample Application**.

3. In the **Employee ID** field, type **50**. Press **Enter**.

   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.

Lesson 3: Use the CustDB client application

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 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.

To browse orders

1. Browsing orders is accomplished in a similar method for each version of the UltraLite client 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.

   To scroll down the list of customers, click **Next**.

2. To scroll up through the list of customers, click **Previous**.

To add an order

1. Adding an order is carried out in a similar way in each version of the UltraLite client application. By adding an order, you have modified the data in your local UltraLite database. This data is not shared with the consolidated database until you synchronize.

   Choose **Order** » **New**.

2. In the **Customer** list, choose **Basements R Us**.

3. In the **Product** list, choose **Screwmaster Drill**. The price of this item is automatically entered in the **Price** field.
4. In the **Quantity** field, type **20**.
5. In the **Discount** field, type **5** (percent) and press **OK**.

To approve, deny, and delete orders

1. 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 are changing the status of it and adding an additional note for the sales person to review. However, the data in the consolidated database is unchanged until you synchronize.

   Approve the order for **Apple Street Builders**.

   a. To locate the customer, click **Previous**.
   b. To approve the order, click **Order** and then **Approve**.
   c. In the **Note** list, choose **Good**.
   d. Press **OK**.

      The order appears with a status of Approved.

2. Deny the order for **Art's Renovations**.

   a. Go to the next order in the list, which is from **Art's Renovations**.
   b. To deny the order, click **Order** and then **Deny**.
   c. In the **Note** list, choose **Discount Is Too High**.
   d. Press **OK**.

   The order appears with a status of Denied.

3. Delete the order for **Awnings R Us**.

   a. Go to the next order in the list, which is from Awnings R Us.
   b. Delete this order by choosing **Order** » **Delete**.

      Click **OK** to confirm the deletion.

      The order is marked as deleted. However, the current data remains in the UltraLite remote until you synchronize changes to the consolidated database.

See also

- "Tables in the CustDB databases" [*MobiLink - Getting Started*]

Lesson 4: Synchronize with the CustDB consolidated database

For synchronization to take place, the MobiLink server must be running. If you have shut down your MobiLink server, you need to restart it. See "Lesson 2: Log in and populate the UltraLite remote database" on page 76.

The synchronization process for the sample application removes approved orders from your database.

You can use Interactive SQL or Sybase Central to connect to the consolidated database and confirm that your changes were synchronized.

To synchronize the UltraLite remote

1. To synchronize your data, from the **File** menu choose **Synchronize**.
2. Confirm that synchronization took place.
   - At the remote database, you can confirm that all required transactions occurred by checking that the approved order for **Apple Street Builders** 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 in the consolidated database.

To confirm the synchronization (Sybase Central)

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. Choose **Connections** » **Connect With SQL Anywhere 12**.
3. Click **ODBC Data Source Name**.
4. Click **Browse** and choose **SQL Anywhere 12 CustDB**.
5. Click **OK**.
6. Click **OK**.
7. Double-click **Tables**.
8. Double-click **ULOrder (DBA)**.
9. Click the **Data** tab and verify that order 5100 is approved, order 5101 is denied, and order 5102 is deleted.

To confirm the synchronization (Interactive SQL)

1. Connect to the consolidated database from Interactive SQL.

a. From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Interactive SQL**.

b. Click **ODBC Data Source Name** and choose **SQL Anywhere 12 CustDB**.

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;
```

Lesson 5: Browse MobiLink synchronization scripts

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.

In addition to the tables, users, and publications, you can also use Sybase Central to browse the synchronization scripts that are stored in the consolidated database. Sybase Central is the primary tool for adding these 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.

For more information about the synchronization logic used in CustDB, see "Synchronization logic source code" [*MobiLink - Getting Started*].

For more information about the implementation of synchronization in CustDB, see "Synchronization design" [*MobiLink - Getting Started*].

To browse the synchronization scripts

1. From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.

2. In the left pane of **Sybase Central**, expand your MobiLink project name, then expand the consolidated database you want to work with. You are connected to the consolidated database based

on the connection information that was provided when you added the consolidated database to your project..

3. Click **ODBC Data Source Name**.
4. Click **Browse** and choose **SQL Anywhere 12 CustDB**.
5. Click **OK**.
6. Click **OK**.
7. Double-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 the synchronization events, you can control the actions that are carried out during synchronization.
8. Click **Synchronized Tables**.
9. In the right pane, double-click **ULCustomer**.

   A set of scripts specific to this table, and their corresponding events appears. These scripts control the way that data in the ULCustomer table is synchronized with the remote databases.

See also

- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "UltraLite clients" on page 93
- "Connection scripts" [*MobiLink - Server Administration*]
- "Table scripts" [*MobiLink - Server Administration*]

Build your own application

Use one of the supported interfaces to build your own application. For more information, see:

- UltraLite C++: "Tutorial: Build an application using the C++ API" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "Tutorial: Build an UltraLite.NET application" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "Tutorial: A sample application for M-Business Anywhere" [*UltraLite - M-Business Anywhere Programming*]

UltraLite performance and optimization

UltraLite provides excellent SQL query performance. Index scans, direct page scans, and temporary tables are internal optimization techniques that help you to achieve the most from the product. However, you can further tune these features, depending on the results of any query performance tests you run.

Using index scans

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. An index is a database object. The index is maintained automatically by UltraLite after it has been created. You can create one or more indexes to improve the performance of your queries, or, depending on the type of index you create, 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. So, when you use them strategically, indexes can greatly improve the performance of searches on the indexed column(s).

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[1]	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[2]	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.

[1] A foreign key can reference either a primary key or a unique key.

[2] Also known as a unique constraint.

Performance tips

- 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
- When creating a composite index, the first column of the index should be the one that is used most often by the predicate in your query.

- Ensure 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).

See also

- "Managing temporary tables" on page 88
- "Using direct page scans" on page 89
- "View an UltraLite execution plan" on page 263
- "About composite indexes" on page 62
- "EXPLANATION function [Miscellaneous]" on page 303
- UltraLite C++: "GetPlan method" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "getPlan method" [*UltraLite - M-Business Anywhere Programming*]
- "UltraLite page_size creation parameter" on page 146

Determining 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. See "Managing temporary tables" on page 88.

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)
  ```

Tuning query performance with 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:

- Partly by the maximum hash size property you configure. See "Choosing an optimal hash size" on page 86.
- Partly by 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, note that 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**.

Now, 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 how are now represented with these hash keys:

Anders

Anderse

Anderse

Anderso

Now, only two of the four row values would need to be unpacked and evaluated, rather than all four.

See also

- “UltraLite max_hash_size creation parameter” on page 143
- “Choosing an optimal hash size” on page 86
- “UltraLite performance and optimization” on page 81
- “Adding UltraLite indexes” on page 64

Choosing an optimal hash size

The UltraLite default maximum hash size of 4 bytes was chosen to suit 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. You need to observe how various hash sizes affect the application and query performance, in addition to the changes in database size itself.

The data type

If you want 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 really needs to, and it never exceeds the maximum hash size you specify. UltraLite always use a smaller hash size if the column type does not use the full byte limit.

Data type	Bytes used to hash the entire value
FLOAT, DOUBLE, and REAL	Not hashed.
BIT and TINYINT	1
SMALL INT and SHORT	2
INTEGER, LONG, and DATE	4
DATETIME, TIME, TIMESTAMP, and BIG	8
CHAR and VARCHAR	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. 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.

Data type	Bytes used to hash the entire value
BINARY	The maximum hash size in bytes must match the declared size of the column. For example, if you declare a column BINARY(30), the required size is 30 bytes.
UUID	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 occurs:

- 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 may render the hash ineffective if you choose a size that only hashes prefixes. In this case, you need to choose a size that ensures more than just the common prefix is hashed. If the common prefix is long, you should 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. The more pages required 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	Page size	Hash size	Number of entries	Pages required
Table C	4 KB	32 bytes	1200 entries	11 pages

See also

- "UltraLite max_hash_size creation parameter" on page 143
- "UltraLite performance and optimization" on page 81
- "Adding UltraLite indexes" on page 64
- "Data types in UltraLite" on page 230

Setting the maximum hash size

You can set the maximum hash size in two ways:

- To store a database *default* for the maximum size, you can 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.
- If you want to *override* the default, you can set a specific hash size when you create a new index. Do one of the following:
  - In Sybase 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.

See also

- "CREATE INDEX statement [UltraLite] [UltraLiteJ]" on page 376
- "CREATE TABLE statement [UltraLite] [UltraLiteJ]" on page 384

Managing temporary tables

In general, the optimizer always 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.

It is difficult to anticipate whether an index you have created avoids the necessity for a temporary table. Therefore, you should always check the plans for a query to ensure the indexes you have created are actually being used by the UltraLite query optimizer.

See also

- "UltraLite temporary tables" on page 11
- "Determining the access method used by the optimizer" on page 83
- "Reading UltraLite execution plans" on page 264

Using 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 pre-existing 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. On the other hand, if order is not important, you can omit the ORDER BY clause to improve query performance.

Note
You cannot use direct page scans if you are using the Table API to program your application.

You can check to see when UltraLite scans a page directly or which index was used to return results. See "Determining the access method used by the optimizer" on page 83.

Reverting to primary key index order

In version 10.0.0 and earlier of UltraLite, the primary key was used to return results when no other index was used by the UltraLite optimizer. As a result, rows were ordered according to the order of the primary key index.

If your results must be ordered by primary key, you should re-write your queries to include the ORDER BY clause.

Tip
It is recommended that you use the ORDER BY clause whenever possible.

See also

- "Using index scans" on page 82
- "SELECT statement [UltraLite] [UltraLiteJ]" on page 402

Flushing 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.

See also

- "UltraLite COMMIT_FLUSH connection parameter" on page 170
- "UltraLite commit_flush_count option [temporary]" on page 163
- "UltraLite commit_flush_timeout option [temporary]" on page 164
- "CHECKPOINT statement [UltraLite]" on page 375
- UltraLite for embedded SQL: "ULCheckpoint method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "Checkpoint method" [*UltraLite - C and C++ Programming*]

How database encryption and obfuscation affect performance

You can encrypt the database to increase the security of the information stored in UltraLite. However, you should note that there is an increase in overhead of between 5-10% as a result, resulting in decreased performance. The exact 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 see any difference at all. To determine what the optimal cache size for your scenario is, you can graph the database performance with benchmark tests.

Stressing the cache

You can benchmark test different cache sizes and watch for performance to change abruptly. Your cache should be large enough to have a good working set of pages. Consider the following ideas to help you stress the cache:

- Create multiple indexes on the table and add foreign keys.
- Insert rows randomly (something other than the index order).
- Create large rows, at least 25% of the database page size.
- Set the index hash to something other than 0. This increased size also increases the page accesses needed.
- Start graphing performance based on the smallest cache size. For example, 256 KB on Windows NT (the smallest allowed cache for this platform) or 64 KB on all other platforms.

If you find that increasing the cache does not improve the performance of an encrypted database, consider obfuscating the data rather than encrypting it. Obfuscation can yield better performance while still offering some security benefits; the obfuscation algorithm uses less code compared to strong encryption, and performs fewer computations. Simple encryption performance should only be marginally slower than no encryption at all. However, your security requirements must ultimately dictate whether you choose to use strong encryption or not.

See also

- "UltraLite performance and optimization" on page 81
- "UltraLite page_size creation parameter" on page 146
- "UltraLite fips creation parameter" on page 142
- "UltraLite CACHE_SIZE connection parameter" on page 167
- "CREATE INDEX statement [UltraLite] [UltraLiteJ]" on page 376

UltraLite as a MobiLink client

This section contains material that describes how to set up and run UltraLite clients for MobiLink synchronization.

UltraLite clients

The UltraLite runtime and engine both include a built-in bi-directional synchronization client. This built-in client means that all data in an UltraLite database is synchronized automatically 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.

For more information about UltraLite, see "Introducing UltraLite" on page 1. For information about how to use SQL Anywhere databases as MobiLink clients, see "SQL Anywhere clients" [*MobiLink - Client Administration*].

Built-in UltraLite synchronization features

UltraLite contains MobiLink synchronization technology in the data management layer for UltraLite. 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.

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

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. See "UltraLite row states" on page 12.

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.

Customizing UltraLite client synchronization behavior

Adding custom synchronization support to UltraLite can involve up to three 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.

  See "Maintaining unique primary keys" [*MobiLink - Server Administration*] and "Primary key uniqueness in UltraLite" on page 95.

- **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. See "Oracle consolidated database" [*MobiLink - Server Administration*] and "UltraLite precision creation parameter" on page 148.

- **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. See "Designing synchronization in UltraLite" on page 99.

  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. See "Adding synchronization to your UltraLite application" on page 104.

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.

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

See also

- "MobiLink consolidated databases" [*MobiLink - Server Administration*]
- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "Direct row handling" [*MobiLink - Server Administration*]
- "Partitioning rows among remote databases" [*MobiLink - Server Administration*]

Primary key uniqueness in UltraLite

UltraLite can maintain primary key uniqueness using any of the techniques supported by MobiLink. See "Maintaining unique primary keys" [*MobiLink - Server Administration*].

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 .

See also

- "UltraLite global_database_id option" on page 165

Using GLOBAL AUTOINCREMENT in UltraLite

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

To declare GLOBAL AUTOINCREMENT columns in your UltraLite database

1. Assign each copy of the database a unique global ID number.

   The global_database_id database option sets the value in your UltraLite database. When deploying UltraLite, you must assign a different identification number to each database. See "UltraLite global_database_id option" on page 165.

2. 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 $pn + 1$. 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 $p(n + 1)$, 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 $pn + 1$ or greater than $p(n + 1)$. 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 negative values, the values UltraLite chooses for GLOBAL AUTOINCREMENT columns 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.

3. 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 method 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. See "Using primary key pools" [*SQL Remote*].

Tip
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.

See also

- "Overriding partition sizes for autoincremented columns" on page 98
- "UltraLite global_database_id option" on page 165
- UltraLite.NET: "Connection property" [*UltraLite - .NET Programming*]
- UltraLite C/C++: "Synchronize method" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "setDatabaseID method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for embedded SQL: "ULSetDatabaseID method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "GlobalAutoIncrementUsage property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "getGlobalAutoIncrementUsage method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for embedded SQL: "ULGlobalAutoincUsage method" [*UltraLite - C and C++ Programming*]
- UltraLiteJ: "setDatabaseId method" [*UltraLiteJ*]
- UltraLiteJ: "getDatabaseId method" [*UltraLiteJ*]

Determining the most recently 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. See "GetLastIdentity method" [*UltraLite - C and C++ Programming*].
- **UltraLite.NET** Use the LastIdentity property on the ULConnection class. See "LastIdentity property" [*UltraLite - .NET Programming*].
- **UltraLite for M-Business Anywhere** Use the GetLastIdentity method on the Connection class. See "getLastIdentity method" [*UltraLite - M-Business Anywhere Programming*].
- **API UltraLiteJ** Use the getlastidentity method on the Connection interface. See "getLastIdentity method" [*UltraLiteJ*].

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.

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.

Overriding partition sizes for autoincremented columns

The partition size is any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition will rarely, if ever, 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.

To override UltraLite partition values (Sybase Central)

1. Connect to the UltraLite database.
2. Right-click the selected column and choose **Properties**.
3. Click the **Value** tab.
4. Enter any positive integer in the **Partition Size** field.

To declare autoincrement columns in UltraLite (SQL)

1. Connect to the UltraLite database.
2. Execute a CREATE TABLE or ALTER TABLE statement with a DEFAULT GLOBAL AUTOINCREMENT clause with the partition size specified in parentheses. See “CREATE TABLE statement [UltraLite] [UltraLiteJ]” on page 384 and “ALTER TABLE statement [UltraLite] [UltraLiteJ]” on page 371.

   For example, the following statement creates 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. A partition size of 5000 is required for this table.

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


See also

- UltraLite.NET: "GetColumnPartitionSize method" [*UltraLite - .NET Programming*]
- UltraLite C/C++: "GetGlobalAutoincPartitionSize method" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "getColumnPartitionSize method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for embedded SQL: "ULGlobalAutoincUsage method" [*UltraLite - C and C++ Programming*]
- API UltraLiteJ: "getLastIdentity method" [*UltraLiteJ*]

Designing synchronization in UltraLite

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 yourself 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 synchronization behavior you require.

Design question	If you answer yes, use the following
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. See "Non-synchronizing tables in UltraLite" on page 101.
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. See "Allsync tables in UltraLite" on page 102.

Design question	If you answer yes, use the following
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?	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. Multiple publications can address priority issues that require certain UltraLite data be uploaded before others. See "Publications in UltraLite" on page 102.
Do you require 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. See "Table order in UltraLite" on page 103.

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. See "UltraLite synchronization parameters and network protocol options" on page 110.
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. See "Deploy UltraLite with TLS-enabled synchronization" on page 47.

See also

- "The synchronization process" [*MobiLink - Getting Started*]
- "The upload and the download" [*MobiLink - Getting Started*]

Non-synchronizing tables in UltraLite

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.

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. See "Publications in UltraLite" on page 102.

Allsync tables in UltraLite

By creating 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.

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.

Publications in UltraLite

Publications define a set of articles that describe the data to be synchronized. Each article 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) if you want 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.

Maximum user publications
The maximum number of user publications in UltraLite is 63.

Adding publications

You can add publications to an UltraLite database with Sybase Central, or using SQL. For UltraLite synchronization, each article in a publication may include either a complete table, or may include a WHERE clause.

Notes
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.

To publish data from an UltraLite database (Sybase Central)

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

To publish data from an UltraLite database (SQL)

1. Connect to the UltraLite database.
2. Execute a CREATE PUBLICATION statement that specifies the name of the new publication and the tables you want to publish.

See also

- “Working with UltraLite publications” on page 65
- “Download Only synchronization parameter” on page 115
- “Upload Only synchronization parameter” on page 130
- “Additional Parameters synchronization parameter” on page 111
- “Publishing data” [*MobiLink - Client Administration*]
- “CREATE PUBLICATION statement [UltraLite] [UltraLiteJ]” on page 378
- “Writing synchronization scripts” [*MobiLink - Server Administration*]

Table order in UltraLite

By setting the Table Order synchronization parameter you can control the order of synchronization operations. If you want to specify a table order for synchronization, you can use the Table Order

parameter programmatically or as part of the ulsync utility during testing. The Table Order parameter specifies the order of tables that are to be uploaded. See "Additional Parameters synchronization parameter" on page 111.

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.

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 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, set one of the foreign keys to **check on commit**. 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.

See also

- "Referential integrity and synchronization" [*MobiLink - Getting Started*]

Adding synchronization to 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, Windows Mobile devices also support ActiveSync synchronization.

Defining the connection

Each UltraLite remote 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. If you require one-way synchronizations only, 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 SQLE_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 disable_concurrency synchronization parameter.

To add synchronization code to your UltraLite application

1. Supply the necessary synchronization parameters and protocol options you require for the session as fields of a synchronization information structure.

   For example, using the C/C++ API, you add synchronization to the UltraLite 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 = ULSocketStream();
          // 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 function. These functions return a boolean 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 ActiveSync synchronization, you must catch the synchronization message from the ActiveSync provider and use the DoSync function to call ULSynchronize.

3. Use an observer callback function if you want to report the progress of the synchronization to the user.

Tip
If you have an environment where DLLs fail either because the DLL is very large or the network connection is unreliable, you may want to implement resumable downloads. See "Handling failed downloads" [*MobiLink - Server Administration*] and "Resuming failed downloads" [*MobiLink - Server Administration*].

See also

- "Using ActiveSync with UltraLite on Windows Mobile" on page 108
- "Upload-only and download-only synchronizations" [*MobiLink - Server Administration*]
- "The upload and the download" [*MobiLink - Getting Started*]
- "UltraLite synchronization parameters and network protocol options" on page 110
- UltraLite.NET: "Synchronization in UltraLite applications" [*UltraLite - .NET Programming*]
- UltraLite C/C++: "Synchronizing data" [*UltraLite - C and C++ Programming*]
- UltraLite C/C++: "Adding ActiveSync synchronization to your application" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Synchronizing data" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for embedded SQL: "Adding synchronization to your application" [*UltraLite - C and C++ Programming*]
- API UltraLiteJ: "Using UltraLiteJ as a MobiLink client" [*UltraLiteJ*]

Using MobiLink file transfers

UltraLite supports the ability to transfer files with the MobiLink server. M-Business Anywhere does not need this functionality because it has its own mechanism for file deployments or transfers (called **channel synchronization**).

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 you require. 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 mlfiletransfer 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 mlfiletransfer 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 Windows Mobile, if the destination is NULL, the file is stored in the root directory (\ ).

     The file name must follow file name conventions for Windows Mobile. See "Windows Mobile" on page 37.

   - 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. See "Desktop" on page 37.

   - On iPhone, 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.

   - On BlackBerry, set the location using the FileTransfer object.

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 stream you require. These are the same parameters supported by UltraLite for MobiLink synchronization. See "UltraLite synchronization parameters and network protocol options" on page 110.

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, except for UltraLiteJ which supports only HTTP.

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.

See also

- "-ftr mlsrv12 option" [*MobiLink - Server Administration*]
- "MobiLink File Transfer utility (mlfiletransfer)" [*MobiLink - Client Administration*]
- UltraLite for C/C++: "MLFileDownload method" [*UltraLite - C and C++ Programming*] and "MLFileUpload method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "ULFileTransfer class" [*UltraLite - .NET Programming*]
- UltraLite for M-Business: Not supported
- API UltraLiteJ: "FileTransfer interface" [*UltraLiteJ*]

Using ActiveSync with UltraLite on Windows Mobile

While you can synchronize data from a Windows Mobile device over an Ethernet or Wi-Fi connection, this section describes how to configure your desktop and device to use ActiveSync synchronization so that your UltraLite database is synchronized at the same time as other ActiveSync operations. If you want to synchronize directly using one of the other alternative methods, you need to program your application to do so using an appropriate synchronize function.

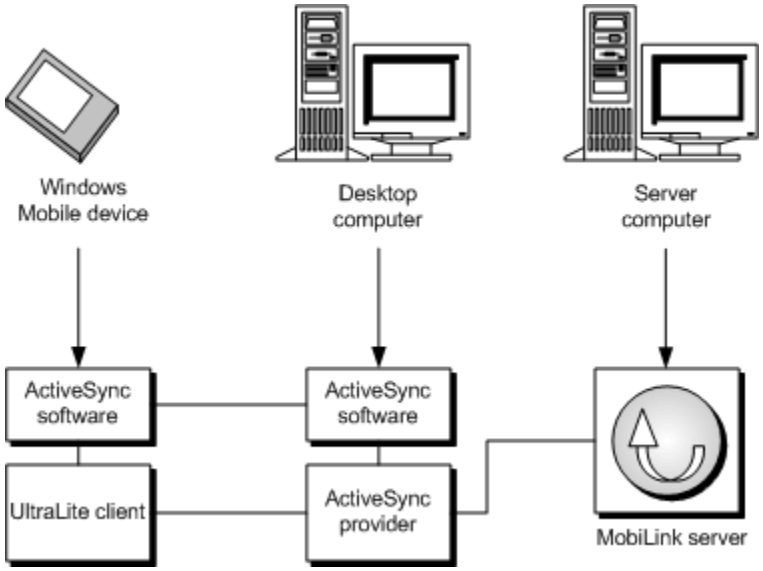
To use ActiveSync initiated synchronization requires that you:

- Register all applications that need to use ActiveSync initiated synchronization with ActiveSync.
- Have the ActiveSync provider installed on your desktop, and deployed to your device.

  To determine which platforms the provider is supported on, see http://www.sybase.com/detail?id=1002288.

The ActiveSync architecture

The following diagram shows the computing layers required by the ActiveSync architecture.



Notice that you must install the ActiveSync provider on your device in addition to your desktop. You can only have a single ActiveSync provider on a single computer. However, if you have more than one UltraLite application installed on a Windows Mobile device, you can register them with the same provider so they are synchronized simultaneously.

See also

- UltraLite C/C++ : "Adding ActiveSync synchronization to your application" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Adding synchronization to your application" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Synchronizing data" [*UltraLite - M-Business Anywhere Programming*]

ActiveSync synchronization overview

1. ActiveSync begins a synchronization session.

2. The 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 ActiveSync, the application calls ULIsSynchronizeMessage to invoke the database synchronization procedure.

5. Once synchronization is complete, the application calls ULSignalSyncIsComplete 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.

UltraLite synchronization parameters and network protocol options

Synchronization parameters for UltraLite

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. This section describes the effects of the parameters, and provides links to other locations for information about to set them.

Note
The parameters described in this section only apply to UltraLite remote databases. To synchronize SQL Anywhere remote databases, see "MobiLink SQL Anywhere client utility (dbmlsync)" [*MobiLink - Client Administration*].

For API-specific details, see:

- UltraLite for Embedded SQL: "ULSynchronize method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "ULSyncParms class" [*UltraLite - .NET Programming*]
- UltraLite C/C++: "Synchronize method" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "SyncParms constructor" [*UltraLiteJ*]

Required parameters

The following parameters are required:

- **Stream Type** See "Stream Type synchronization parameter" on page 127.
- **User Name** See "User Name synchronization parameter" on page 131.
- **Version** See "Version synchronization parameter" on page 132.

If you do not set these parameters, the synchronization function throws an exception (for example, SQLCode.SQLE_SYNC_INFO_INVALID or its equivalent).

Conflicting parameters

You can specify at most one of these parameters:

- **Download Only** See "Download Only synchronization parameter" on page 115.
- **Ping** See "Ping synchronization parameter" on page 121.

- **Upload Only** See "Upload Only synchronization parameter" on page 130.

If you set more than one of these parameters to true, the synchronization function throws an exception (for example, SQLCode.SQLE_SYNC_INFO_INVALID or its equivalent).

Additional Parameters synchronization parameter

This synchronization parameter allows an application to supply additional parameters that can not be readily specified using any other predefined parameters. Some parameters that are seldom used are specified in this parameter field.

The additional parameters are supplied as a string of keyword=value settings, separated with a semicolon.

Syntax

The syntax varies depending on the API you use. It is not available in UltraLiteJ.

Allowed values

The following properties can be specified as part of the additional parameters setting:

Property name	Description
AllowDownloadDupRows	Prevents errors from being raised when a synchronization encounters downloaded rows with duplicate primary keys. Set this property to **0** to raise errors and roll back the download; otherwise, set to **1** to raise warnings and continue the download. This property is only available in UltraLite C/C++.
CheckpointStore	Adds additional checkpoints of the database during synchronization to limit database growth during the synchronization process. 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.
DisableConcurrency	Disallows database access from other threads during synchronization during the upload phase. Set this property to **0** to allow concurrent database access; otherwise, set to **1**. By default, this property is set to **0**.

Property name	Description
TableOrder	Sets the table order required for priority synchronization if the UltraLite default table ordering is not suitable for your deployment. 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 and there are no foreign key cycles. Quote tables names with either single or double quotes. For example, "Customer,Sales" and 'Customer,Sales' are both supported in UltraLite. 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. The order of tables on the download is the same as those you define for upload. You only need to explicitly set the table order if your UltraLite tables: • Are part of foreign key cycles. You must then list all tables that are part of a cycle. • Have different foreign key relationships in the consolidated database.

See also

- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "AdditionalParms property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "setAdditionalParms method" [*UltraLite - M-Business Anywhere Programming*]

Example

UltraLite for C/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"
);
```

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. See "Number of Authentication Parameters parameter" on page 118.

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.

See also

- "Number of Authentication Parameters parameter" on page 118
- "Authentication parameters" [*MobiLink - Server Administration*]
- "authenticate_parameters connection event" [*MobiLink - Server Administration*]
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "AuthenticationParms property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "getAuthenticationParms method" [*UltraLite - M-Business Anywhere Programming*]
- API UltraLiteJ: "setAuthenticationParms method" [*UltraLiteJ*]

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;
```

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. See "authenticate_user connection event" [*MobiLink - Server Administration*].

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.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "AuthStatus property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncResult class" [*UltraLite - M-Business Anywhere Programming*]
- API ULtraLiteJ: "getAuthStatus method" [*UltraLiteJ*]

Example

UltraLite for C/C++ applications can access the parameter as follows:

```
ul_sync_info info;
// ...
returncode = info.auth_status;
```


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.

See also

- "authenticate_user connection event" [*MobiLink - Server Administration*]
- "authenticate_user_hashed connection event" [*MobiLink - Server Administration*]
- "Authentication Status synchronization parameter" on page 113
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "AuthValue property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncResult class" [*UltraLite - M-Business Anywhere Programming*]
- API UltraLiteJ: "getAuthValue method" [*UltraLiteJ*]

Example

UltraLite for C/C++ applications can access the parameter as follows:

```
ul_sync_info info;
// ...
returncode = info.auth_value;
```

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. You can also set this parameter with ulsync.

Default

False

Allowed values

Boolean

Conflicts with

Ping and Upload Only

Remarks

When you have remotes that are synchronized by download-only synchronization, you should regularly do a full synchronization to reduce the amount of log that is scanned by the download-only synchronization. Otherwise, the download-only synchronizations will take an increasingly long time to complete.

For ulsync When download-only synchronization occurs, ulsync does not upload any changes to the data. Instead, it:

- Uploads information about the schema and the value stored in the progress counter.
- Ensures that changes on the remote are not overwritten during download-only synchronization.

ulsync performs these actions by scanning the UltraLite database log to watch for rows with pending operations on the consolidated database. If ulsync detects a conflict, the download is rolled back and the synchronization fails. You must then do a full synchronization (that is an upload and a download) to correct this conflict.

See also

- "Upload Only synchronization parameter" on page 130
- "Synchronization state tracking" on page 93
- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "DownloadOnly property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- API UltraLite J: "setDownloadOnly method" [*UltraLiteJ*]

Examples

ulsync supports this parameter as an extended synchronization parameter:

```
ulsync -c DBF=myuldb.udb
"MobiLinkUid=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;
```

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.

See also

- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "IgnoredRows property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncResult class" [*UltraLite - M-Business Anywhere Programming*]
- API UltraLiteJ: "getIgnoredRows method" [*UltraLiteJ*]

Example

UltraLite for C/C++ applications can access the parameter as follows:

```
ul_sync_info info;
// ...
res = info.ignored_rows;
```

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. It is not available in UltraLiteJ.

Default

False, which indicates that UltraLite rolls back all changes after a failed download.

Allowed values

Boolean

See also

- "Resuming failed downloads" [*MobiLink - Server Administration*]
- "Resume Partial Download synchronization parameter" on page 123
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "KeepPartialDownload property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]

Example

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
// ...
info.keep_partial_download = ul_true;
```

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.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "NewPassword property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "setPassword method" [*UltraLiteJ*]

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" );
```

Number of Authentication Parameters 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.

See also

- "Authentication Parameters synchronization parameter" on page 112
- "authenticate_parameters connection event" [*MobiLink - Server Administration*]
- "Authentication parameters" [*MobiLink - Server Administration*]
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "AuthenticationParms property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]

Example

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
// ...
info.num_auth_parms = 3;
```

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 );
```

The **ul_sync_status** structure is described in "ul_sync_status structure" [*UltraLite - C and C++ Programming*].

Syntax

The syntax varies depending on the API you use.

See also

- "User Data synchronization parameter" on page 131
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "ULSyncProgressListener interface" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "synchronizeWithParm method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "setSyncObserver method" [*UltraLiteJ*]

Example

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
// ...
info.observer=callfunction;
```

Partial Download Retained synchronization parameter

This field is set by a synchronization to indicate whether UltraLite applied those changes that were downloaded 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. Not available in UltraLiteJ.

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. See "Keep Partial Download synchronization parameter" on page 117.

See also

- "Resuming failed downloads" [*MobiLink - Server Administration*]
- "Resume Partial Download synchronization parameter" on page 123
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "PartialDownloadRetained property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncResult class" [*UltraLite - M-Business Anywhere Programming*]

Example

Access the parameter as follows:

```
ul_sync_info info;
// ...
returncode=info.partial_download_retained;
```

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. See "User Name synchronization parameter" on page 131.

If the MobiLink client already has a password, use the New Password parameter to change it. See "New Password synchronization parameter" on page 118.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "Password property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "setPassword method" [*UltraLiteJ*]

Examples

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" );
```

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

See also

- "-pi dbmlsync option" [*MobiLink - Client Administration*]
- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "PingOnly property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ:"setPingOnly method" [*UltraLiteJ*]

Examples

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;
```

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.

See also

- "Publications in UltraLite" on page 102
- "Working with UltraLite publications" on page 65
- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "SYNC_ALL_DB field" [*UltraLite - .NET Programming*] and "SYNC_ALL_PUBS field" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "PublicationSchema class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "setPublications method" [*UltraLiteJ*]

Example

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb
"MobiLinkUid=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" );
```

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. Not available in UltraLiteJ.

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.

See also

- "Resuming failed downloads" [*MobiLink - Server Administration*]
- "Keep Partial Download synchronization parameter" on page 117
- "Partial Download Retained synchronization parameter" on page 120
- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "ResumePartialDownload property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]

Example

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
// ...
info.resume_partial_download = ul_true;
```

Send Column Names synchronization parameter

Specifies that column names should be sent in the upload.

Syntax

The syntax varies depending on the API you use. You can also set this parameter with ulsync.

Default

False

Allowed values

Boolean

Remarks

This option is used by the MobiLink server for direct row handling. When using direct row handling, you should enable this option. Otherwise, it has no effect. See "Direct row handling" [*MobiLink - Server Administration*].

See also

- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "SendColumnNames property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "setSendColumnNames method" [*UltraLiteJ*]

Examples

ulsync can set this parameter as an extended synchronization parameter as follows:


```
ulsync -c DBF=myuldb.udb
"MobiLinkUid=remoteA;ScriptVersion=2;SendColumnNames=true;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
// ...
info.send_column_names = ul_true;
```

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

See also

- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "SendDownloadAck property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "setAcknowledgeDownload method" [*UltraLiteJ*]

Examples

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb
"MobiLinkUid=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;
```

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/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 a listing of error numbers, see "MobiLink communication error messages" [*Error Messages*]. For the error code suffixes, see *install-dir\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, this means that the MobiLink server is not running or is not listening on the specified port. See http://msdn2.microsoft.com/en-us/library/ms740668.aspx.
- **error_string** An application-provided error message. The string may or may not be empty. A non-empty error_string provides information in addition to the stream_error_code. For instance, for a write error (error code 9) the error string is a number showing how many bytes it was trying to write.
- **error_string_length** Deprecated. The size of the error string buffer.

Remarks

UltraLite applications other than the UltraLite C++ Component receive communications error information as part of the Sync Result parameter. See "Sync Result synchronization parameter" on page 129.

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 *install-dir\SDK\Include\sserror.h*.

Check for SQLE_COMMUNICATIONS_ERROR:

```
Connection conn;
ul_sync_info info;
```



```
...
conn.InitSynchInfo( &info );
info.stream_error.error_string = error_buff;
info.stream_error.error_string_length =
                    sizeof( error_buff );
if( !conn.Synchronize( &synch_info ) ){
    if( SQLCODE == SQLE_COMMUNICATIONS_ERROR ){
        printf( error_buff );
    // more error handling here
```

See also

- "ul_sync_info structure" [*UltraLite - C and C++ Programming*]

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. See "Stream Parameters synchronization parameter" on page 128.

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.
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. This protocol is not available in UltraLiteJ.
TLS	Synchronize over TCP/IP with transport-layer security (TLS). TLS secures client/server communications using digital certificates and public-key cryptography.

For a list of supported platforms, see http://www.sybase.com/detail?id=1061806.

See also

- "Certificate Creation utility (createcert)" [*SQL Anywhere Server - Database Administration*]
- "Certificate Viewer utility (viewcert)" [*SQL Anywhere Server - Database Administration*]
- "Transport-layer security" [*SQL Anywhere Server - Database Administration*]
- "UltraLite Synchronization utility (ulsync)" on page 209
- "Network protocol options for UltraLite synchronization streams" on page 133
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "Stream property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]

Example

For UltraLite for C/C++ applications, set the parameter as follows:

```
Connection conn;
ul_sync_info info;
...
conn.InitSynchInfo( &info );
info.stream = "http";
```

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.

See also

- "UltraLite Synchronization utility (ulsync)" on page 209
- "Network protocol options for UltraLite synchronization streams" on page 133
- UltraLite for C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "StreamParms property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "StreamHTTPParms interface" [*UltraLiteJ*]

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" );
```

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. See "Authentication Status synchronization parameter" on page 113.
- **Ignored Rows** Reports the number of ignored rows. See "Ignored Rows synchronization parameter" on page 116.
- **Stream Error information** The Stream Error information includes a Stream Error Code, Stream Error Context, Stream Error ID, and Stream Error System. See "Stream Error synchronization parameter" on page 125.
- **Upload OK** Reports the success or failure of the upload phase. See "Upload OK synchronization parameter" on page 129.

See also

- UltraLite.NET: "ULSyncParms class" [*UltraLite - .NET Programming*]
- UltraLite C/C++: "ul_sync_result structure" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for embedded SQL: "ULGetSyncResult method" [*UltraLite - C and C++ Programming*]
- UltraLiteJ: "SyncResult class" [*UltraLiteJ*]

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.

See also

- UltraLite C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "UploadOK property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncResult class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "isUploadOK method" [*UltraLiteJ*]

Example

UltraLite for C/C++ applications can access the parameter as follows:

```
ul_sync_info info;
// ...
returncode = info.upload_ok;
```

Upload Only synchronization parameter

Indicates that there should be no downloads in the current synchronization, which can save communication time, especially 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.

See also

- "Designing synchronization in UltraLite" on page 99
- "Download Only synchronization parameter" on page 115
- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "UploadOnly property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "isUploadOnly method" [*UltraLiteJ*]

Examples

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb
"MobiLinkUid=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;
```

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.

See also

- "Observer synchronization parameter" on page 119
- UltraLite C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]

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. See "Remote IDs" [*MobiLink - Client Administration*].

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. See "Password synchronization parameter" on page 120.

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.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "UltraLite user authentication" [*MobiLink - Client Administration*]
- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "UserName property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ:"setUserName method" [*UltraLiteJ*]

Examples

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" );
```

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.

See also

- "Script versions" [*MobiLink - Server Administration*]
- "UltraLite Synchronization utility (ulsync)" on page 209
- UltraLite C/C++: "ul_sync_info structure" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "Version property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "SyncParms class" [*UltraLite - M-Business Anywhere Programming*]
- UltraLiteJ: "setVersion method" [*UltraLiteJ*]

Examples

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" );
```

Network protocol options for UltraLite synchronization streams

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. Support is also provided for ActiveSync notification on Windows Mobile.

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 network protocol options provide information such as addressing information (host and port) and protocol-specific information. Refer to the table below to determine which options you can use for the stream type you are using.

For a list of protocol options, see "MobiLink client network protocol option summary" [*MobiLink - Client Administration*].

See also

- "Configuring UltraLite clients to use transport-layer security" [*SQL Anywhere Server - Database Administration*]
- "Deploy UltraLite with TLS-enabled synchronization" on page 47
- "MobiLink client network protocol options" [*MobiLink - Client Administration*]
- "Stream Parameters synchronization parameter" on page 128
- -x option in "UltraLite Synchronization utility (ulsync)" on page 209

Setting the synchronization stream and options

You can provide the information needed to locate the MobiLink server in your application by setting the Stream Parameters parameter. See "Stream Parameters synchronization parameter" on page 128.

For information about including stream parameters in your UltraLite synchronization call, see:

- UltraLite for UltraLite.NET: "StreamParms property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "setStreamParms method" [*UltraLite - M-Business Anywhere Programming*]

UltraLite database reference

This section provides a reference for UltraLite database properties, options, connection parameters, and utilities.

UltraLite creation parameters

Creation parameters are used to configure the UltraLite database when you first create it. You can only change these settings by recreating the database.

You can specify creation parameters when creating a database using the ulinit or ulload utility, and from the supported client interfaces.

Boolean creation parameters are turned on with YES, Y, ON, TRUE, T, or 1, and are turned off with any of NO, N, OFF, FALSE, F, and 0. The parameters are case insensitive.

UltraLite creation parameters are specified in a semicolon separated string when creating a database from a programming interfaces and as a separate command line parameters when using a command line utility. For example:

```
ulinit --case --utf8_encoding=1 test.udb
```

Alternatively, you can specify multiple -c options.

See also

- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- "Accessing creation parameter values" on page 28

UltraLite case creation parameter

Sets the case sensitivity of string comparisons in the UltraLite database. 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.

Syntax

ulinit --case *database.udb*

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. See "Strings in UltraLite" on page 225.

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 Sybase Central, you can set the case sensitivity in any wizard that creates a database. On the **New Database Collation And Character Set** page, select the **Use Case-sensitive String Comparisons** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

UltraLite checksum_level creation parameter

Sets the level of checksum validation for the database.

Syntax

ulinit --checksum_level=*value*

Allowed values

0, 1, 2

Default

0

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, the page has been modified or became corrupted during the storage/retrieval of the page. If a checksum validation fails, 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 iAnywhere.

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.

From Sybase Central, you can configure the use of checksums in any wizard that creates a database. On the **New Database Storage Settings** page, select the **Checksum Level For Database Pages** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- checksum_level database property: “UltraLite database properties” on page 158
- “UltraLite Initialize Database utility (ulinit)” on page 197
- “UltraLite Load XML to Database utility (ulload)” on page 205
- UltraLite for embedded SQL: “ULCreateDatabase method” [*UltraLite - C and C++ Programming*]
- UltraLite for C++: “CreateDatabase method” [*UltraLite - C and C++ Programming*]
- UltraLite.NET: “CreateDatabase method” [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: “createDatabase method” [*UltraLite - M-Business Anywhere Programming*]
- “UltraLite performance and optimization” on page 81
- “UltraLite page_size creation parameter” on page 146
- “Connecting to an UltraLite database” on page 34

UltraLite collation creation parameter

Sets the database collation.

Syntax

ulinit --collation=*value*

Allowed values

String

Default

1252Latin1

Remarks

For a list of supported collations in UltraLite, see "UltraLite supported collations" on page 31.

You can also view a list of supported collations in UltraLite by executing the following command:

```
ulinit -Z
```

From Sybase Central, you can set the collation in any wizard that creates a database. On the **New Database Collation And Character Set** page, choose either the default collation (1252Latin1), or an alternate one from the list.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- "UltraLite character sets" on page 30
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

UltraLite date_format creation parameter

Sets the format for dates retrieved from the database.

Syntax

ulinit --date_format=*value*

Allowed values

String

Default

YYYY-MM-DD (this corresponds to ISO date format specifications)

Remarks

DATE data type values are represented in a format set by the date_format creation parameter. Date values can, however, also be represented by strings. Before the value can be retrieved, it must be assigned to a string.

UltraLite builds a date from date parts. Date parts can include the year, the month, the day of the month, the day of the week, the day of the year, the hour, the minute, and the second (and parts thereof).

ISO (YYYY-MM-DD) is the default date format and order. For example, "7th of January 2006" in this international format is written: 2006-01-07. If you do not want to use the default ISO date format and order, you must specify a different format and order for these date 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. 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.
hh	Two digit hours. A leading zero is not required.
nn	Two digit minutes. A leading zero is not required.
ss[.*ss*..]	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 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.

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.

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 Sybase Central, you can set the date format in any wizard that creates a database. On the **New database creation parameters** page, select the **Date Format** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite date_order creation parameter" on page 141
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

Example

The following table illustrates date_format settings, together with the output from a SELECT CURRENT DATE statement, executed on Thursday May 21, 2001.

date_format syntax used	Result returned
yyyy/mm/dd/ddd	2001/05/21/thu

date_format syntax used	Result returned
jjj	141
mmm yyyy	may 2001
mm-yyyy	05-2001

UltraLite date_order creation parameter

Controls the interpretation of date formats.

Syntax

ulinit --date_order=*value*

Allowed values

MDY, YMD, DMY

Default

YMD (this corresponds to ISO date format specifications)

Remarks

DATE data type values are represented in a format set by the date_format creation parameter. Date values can, however, also be represented by strings. Before the value can be retrieved, it must be assigned to a string.

UltraLite builds a date from date parts. Date parts can include the year, the month, the day of the month, the day of the week, the day of the year, the hour, the minute, and the second (and parts thereof).

ISO (YYYY-MM-DD) is the default date format and order. For example, "7th of January 2006" in this international format is written: 2006-01-07. If you do not want to use the default ISO date format and order, you must specify a different format and order for these date parts.

You cannot change the date order of an existing database. Instead, you must create a new database.

From Sybase Central, you can set the date order in any wizard that creates a database. On the **New database creation parameters** page, select the **Date Order** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite date_format creation parameter" on page 138
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

Example

Different values determine how the date of 10/11/12 is translated:

Syntax used	Translation
MDY	Oct 11 1912
YMD	Nov 12 1910
DMY	Nov 10 1912

UltraLite fips creation parameter

Controls whether the new database should be encrypted using AES or AES_FIPS strong encryption.

Syntax

{**ulinit -a** | **ulload -c** } **--fips=***value* **--key=***value*

Allowed values

Yes (use AES_FIPS), No (use AES)

Default

Yes

Remarks

The only way to change the type of database encryption is to recreate the database with the appropriate fips or obfuscate creation parameter. 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 Sybase Central, you can configure strong encryption in any wizard that creates a database. On the **New Database Storage Settings** page, select the **AES FIPS Algorithm** option. You must also set and confirm the encryption key.

Set this parameter as one of the creation parameters for the create database method on the database manager class.

To deploy a FIPS-enabled database, copy all appropriate libraries for your platform. See "Deploy UltraLite with AES_FIPS database encryption" on page 46.

See also

- "Strong encryption" [*SQL Anywhere Server - Database Administration*]
- "Securing UltraLite databases" on page 32
- "UltraLite obfuscate creation parameter" on page 146
- "UltraLite DBKEY connection parameter" on page 174
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite C/C++: "ULChangeEncryptionKey method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite.NET: "ChangeEncryptionKey method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "changeEncryptionKey method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

UltraLite max_hash_size creation parameter

Sets the maximum default index hash size in bytes.

Syntax

ulinit --max_hash_size=*value*

Allowed values

0 to 32 bytes

Default

4 bytes

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 parameter. 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 new index, you can override the default value with the UltraLite **Create Index Wizard** in Sybase 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 Sybase Central, you can set the maximum hash size in any wizard that creates a database. On the **New Database Storage Settings** page, select the **Maximum Hash Size For Indexes** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite performance and optimization" on page 81
- "Working with UltraLite indexes" on page 61
- "Choosing an optimal hash size" on page 86
- "CREATE INDEX statement [UltraLite] [UltraLiteJ]" on page 376
- "CREATE TABLE statement [UltraLite] [UltraLiteJ]" on page 384
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

UltraLite nearest_century creation parameter

Controls the interpretation of two-digit years in string-to-date conversions.

Syntax

ulinit --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*.

  It is recommended that you store four-digit dates to avoid issues with incorrect conversions. See "Ambiguous date and time conversions" [*SQL Anywhere Server - SQL Reference*].

- **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 Sybase Central, you can configure the nearest century setting in any wizard that creates a database. On the **New database creation parameters** page, select the **Nearest Century** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

UltraLite obfuscate creation parameter

Controls obfuscation of data in the database. Obfuscation is a form of simple encryption.

Syntax

{**ulinit -a** | **ulload -c** } **obfuscate=***value*

Allowed values

Boolean.

Default

0 (databases are not obfuscated)

Remarks

Simple encryption is equivalent to obfuscation and makes it more difficult for someone using a disk utility to look at the file to decipher the data in your database. Simple encryption does not require a key to encrypt the database.

If you want to make the database inaccessible without the correct encryption key, you must use strong encryption. See "UltraLite fips creation parameter" on page 142.

From Sybase Central, you can set obfuscation in any wizard that creates a database. On the **New Database Storage Settings** page, select the **Use Simple Encryption (Obfuscation)** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "Simple encryption" [*SQL Anywhere Server - Database Administration*]
- "Securing UltraLite databases" on page 32
- "UltraLite fips creation parameter" on page 142
- "UltraLite DBKEY connection parameter" on page 174
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++ : "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

UltraLite page_size creation parameter

Defines the database page size in kilobytes.

Syntax

{**ulinit -a** | **ulload -c** } **--page_size=***size*]

Allowed values

1, 2, 4, 8, 16

Default

4

Remarks

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. See "Row packing and table definitions" on page 53.
- **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. When your cache cannot hold enough pages, performance suffers as UltraLite begins swapping frequently-used pages to disk. See "UltraLite CACHE_SIZE connection parameter" on page 167.
- **Index entries** Page size also affects indexes. The larger the database page, the more index entries it can hold. See "Working with UltraLite indexes" on page 61.
- **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 Sybase Central, you can set the page size in any wizard that creates a database. On the New Database Storage Settings page, select the appropriate byte value.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "Row packing and table definitions" on page 53
- "UltraLite case creation parameter" on page 135
- "UltraLite CACHE_SIZE connection parameter" on page 167
- "UltraLite RESERVE_SIZE connection parameter" on page 181
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

Example

To set the page size of the database to 8 KB, specify **page_size=8k** or **page_size=8192**:

```
ulinit test.udb -a --page_size=8k
```

UltraLite precision creation parameter

Specifies the maximum number of digits in decimal point arithmetic results.

Syntax

ulinit 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 Sybase Central, you can set the precision in any wizard that creates a database. On the **New database creation parameters** page, select the **Precision** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

UltraLite scale creation parameter

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

Syntax

ulinit --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 Sybase Central, you can set the scale in any wizard that creates a database. On the **New database creation parameters** page, select the **Scale** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite precision creation parameter" on page 148
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

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.

UltraLite time_format creation parameter

Sets the format for times retrieved from the database.

Syntax

ulinit --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 parameter. 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.

ISO (HH:MM:SS) is the default time format. For example, "midnight" in this international format is written: 00:00:00. If you do not want to use the default ISO time format, 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*).
SS[.s...]	Two digit seconds plus optional fraction.

You cannot change the time format 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.

You can 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.

From Sybase Central, you can set the time format in any wizard that creates a database. On the **New database creation parameters** page, select the **Time Format** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite timestamp_format creation parameter" on page 152
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

Example

If a transaction was executed at 3:30 P.M. and you used the default time_format syntax of *HH*:*NN*:*SS.sss*, the result would be:

15:30:55.0

UltraLite timestamp_format creation parameter

Sets the format for timestamps that are retrieved from the database.

Syntax

ulinit --timestamp_format=*value*

Allowed values

String

Default

YYYY-MM-DD HH:*NN*:*SS.SSS*

Remarks

UltraLite creates a timestamp from date and time parts that you set with the date_format and time_format creation parameters. Together, date and time total seven parts (year, month, day, hour, minute, second, and millisecond).

Timestamp values can also be represented by strings. Before it can be retrieved, a timestamp value must be assigned to a string variable.

Typically timestamp columns ensure that data integrity is maintained when synchronizing with a consolidated database. Timestamps help identify when concurrent data updates have occurred among multiple remote databases by tracking the last time that each user synchronized.

Tip
Ensure that the consolidated database and the UltraLite remote maintain timestamps and timestamp increments to the same resolution. By setting the these creation parameters to match that of the consolidated database, you can help avoid spurious inequalities.

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. 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.
HH	Two digit hours. A leading zero is not required.
NN	Two digit minutes. A leading zero is not required.
SS[.*ss*..]	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 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*), 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.

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 Sybase Central, you can set the timestamp format in any wizard that creates a database. On the **New database creation parameters** page, select the **Timestamp Format** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite timestamp_increment creation parameter" on page 154
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28
- "Timestamp-based downloads" [*MobiLink - Server Administration*]
- "Concurrency in UltraLite" on page 11

Example

If a transaction was executed on Friday May 12, 2006 at 3:30 PM and you used the default timestamp_format syntax of *YYYY-MM-DD HH:NN:SS.SSS*, the result would be:

2006-05-12 15:30:55.0

UltraLite timestamp_increment creation parameter

Limits the resolution of timestamp values. As timestamps are inserted into the database, UltraLite truncates them to match this increment.

Syntax

ulinit timestamp_increment=*value*

Allowed values

1 to 60000000 microseconds

Default

1 microsecond

Remarks

1000000 microseconds equals 1 second.

You cannot change the timestamp increment 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 Sybase Central, you can set the timestamp increment in any wizard that creates a database. On the **New database creation parameters** page, select the **Timestamp Increment** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite timestamp_format creation parameter" on page 152
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for embedded SQL: "ULCreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28
- "Timestamp-based downloads" [*MobiLink - Server Administration*]
- "Concurrency in UltraLite" on page 11

Example

To store a value such as '2000/12/05 10:50:53:700', set this creation parameter to 100000. This value truncates the timestamp after the first decimal place in the seconds component.

UltraLite timestamp_with_time_zone_format creation parameter

Sets the format for TIMESTAMP WITH TIME ZONE values retrieved from the database.

Syntax

ulinit --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 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
SS.SSSSSS	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 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, 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.

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. See "date_order option" [*SQL Anywhere Server - Database Administration*].

See also

- "TIMESTAMP WITH TIME ZONE data type" [*SQL Anywhere Server - SQL Reference*]

UltraLite utf8_encoding creation parameter

Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode.

Syntax

ulload - c --utf8_encoding

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. This means that 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, Windows Mobile and iPhone) 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 Sybase Central, you can choose UTF-8 encoding in any wizard that creates a database. On the **New database collation and character set** page, select the **Yes, use UTF-8 as the database character set** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- "UltraLite platform requirements for character set encoding" on page 31
- "UltraLite character sets" on page 30
- "UltraLite Initialize Database utility (ulinit)" on page 197
- "UltraLite Load XML to Database utility (ulload)" on page 205
- UltraLite for C++: "CreateDatabase method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "CreateDatabase method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "createDatabase method" [*UltraLite - M-Business Anywhere Programming*]
- "Accessing creation parameter values" on page 28

UltraLite database properties

Property	Description
case	Returns the status of the case sensitivity feature. Returns On if the database is case sensitive. Otherwise, it returns Off. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite case creation parameter" on page 135.
char_set	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. See also: • "UltraLite utf8_encoding creation parameter" on page 157 • "UltraLite collation creation parameter" on page 137 The value of this property is set when the database is created, and can only be changed by creating a new database.

Property	Description
checksum_level	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 value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite checksum_level creation parameter" on page 136.
collation	Returns the name of the database's collation sequence. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite collation creation parameter" on page 137.
commit_flush_count	Returns the value of the commit_flush_count option that sets a commit count threshold. See "UltraLite commit_flush_count option [temporary]" on page 163.
commit_flush_timeout	Returns the value of the commit_flush_timeout option that sets a time interval threshold. See "UltraLite commit_flush_timeout option [temporary]" on page 164.
conn_count	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.
date_format	Returns the date format the database uses for string conversions. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite date_format creation parameter" on page 138.
date_order	Returns the date order the database uses for string conversions. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite date_order creation parameter" on page 141.

Property	Description
encryption	Returns the type of database encryption, one of None, Simple, AES, or AES_FIPS. The encryption used by the database is determined by whether or not you have configured strong encryption (AES or AES_FIPS) and the DBKEY creation parameter, or obfuscation (simple encryption). 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. See: ● UltraLite C/C++: "ULChangeEncryptionKey method" [*UltraLite - C and C++ Programming*] ● UltraLite.NET: "ChangeEncryptionKey method" [*UltraLite - .NET Programming*] ● UltraLite for M-Business Anywhere: "changeEncryptionKey method" [*UltraLite - M-Business Anywhere Programming*] ● "Securing UltraLite databases" on page 32 ● "UltraLite fips creation parameter" on page 142 ● "UltraLite obfuscate creation parameter" on page 146 ● "UltraLite DBKEY connection parameter" on page 174
file	Returns the name of the database root file for the current connection, the including path. This is the value specified in the DBF connection parameter value. See: ● UltraLite C/C++: "GetDatabaseProperty method" [*UltraLite - C and C++ Programming*] ● UltraLite.NET: "GetDatabaseProperty method" [*UltraLite - .NET Programming*] ● UltraLite for M-Business Anywhere: "getDatabaseProperty method" [*UltraLite - M-Business Anywhere Programming*] ● "UltraLite DBF connection parameter" on page 172
global_database_id	Returns the value of the global_database_id option used for global autoincrement columns. See "UltraLite global_database_id option" on page 165.
max_hash_size	Returns the default number of maximum bytes to use for index hashing. This property can be set on a per-index basis. See "UltraLite max_hash_size creation parameter" on page 143.

Property	Description
ml_remote_id	Returns the value of the ml_remote_id option that uniquely identifies the database for MobiLink synchronization. See "UltraLite ml_remote_id option" on page 166.
name	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. See also: ● "UltraLite DBN connection parameter" on page 174 ● "UltraLite DBF connection parameter" on page 172
nearest_century	Returns the nearest century the database uses for string conversions. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite nearest_century creation parameter" on page 144.
page_size	Returns the page size of the database, in bytes. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite page_size creation parameter" on page 146.
precision	Returns the floating-point precision the database uses for string conversions. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite precision creation parameter" on page 148.
scale	Returns the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION during string conversions by the database. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite scale creation parameter" on page 149.
time_format	Returns the time format the database uses for string conversions. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite time_format creation parameter" on page 150.
timestamp_format	Returns the timestamp format the database uses for string conversions. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite timestamp_format creation parameter" on page 152.

Property	Description
timestamp_increment	Returns the minimum difference between two unique time-stamps, in microseconds. The value of this property is set when the database is created, and can only be changed by creating a new database. See "UltraLite timestamp_increment creation parameter" on page 154.

Accessing UltraLite database properties

UltraLite provides a set of properties that you can retrieve for a database.

You can change the settings of any database property that does not correspond to a database creation parameter.

To browse UltraLite database properties (Sybase Central)

1. Connect to the database.
2. Right-click the database and choose **Properties**.

   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.

To get the value of a database property (C/C++)

- In C/C++, call the GetDatabaseProperty function.

  For example, to get the value of the conn_count property, call:

  ```
  GetDatabaseProperty( ul_database_property_id conn_count )
  ```

  To get the value of the char_set property, call:

  ```
  GetDatabaseProperty( ul_database_property_id char_set )
  ```

See also

- UltraLite C/C++: "GetDatabaseProperty method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "GetDatabaseProperty method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "getDatabaseProperty method" [*UltraLite - M-Business Anywhere Programming*]

UltraLite database options

Options are used to configure database behavior. Database options can be set or modified at any time. In UltraLite, options can be persistent or temporary. Persistent options are stored in the database in the sysuldata system table. Temporary option settings only persist while the database is running.

Option values are set by using the SET OPTION statement. For example, the following statement sets the global_database_id option to 100:

```
SET OPTION global_database_id=100;
```

You can obtain the current setting of a database option by querying the value of the corresponding database property or by using the appropriate get database property method. For example, to obtain the current setting of the commit_flush_timeout database option, execute the following SQL statement:

```
SELECT DB_PROPERTY ( 'commit_flush_timeout' );
```

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.

You must set this option each time you start the database if it is required.

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.

See also

- "Flushing single or grouped transactions" on page 89
- "UltraLite commit_flush_timeout option [temporary]" on page 164
- "UltraLite COMMIT_FLUSH connection parameter" on page 170
- "SET OPTION statement [UltraLite] [UltraLiteJ]" on page 404
- UltraLite for C/C++: "SetDatabaseOption method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "ULSetDatabaseOptionString method" [*UltraLite - C and C++ Programming*]

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.

You must set this option each time you start database, if it is required.

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.

See also

- "UltraLite commit_flush_count option [temporary]" on page 163
- "UltraLite COMMIT_FLUSH connection parameter" on page 170
- "SET OPTION statement [UltraLite] [UltraLiteJ]" on page 404
- UltraLite for C/C++: "SetDatabaseOption method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "ULSetDatabaseOptionString method" [*UltraLite - C and C++ Programming*]

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 $pn + 1$ 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. See "Using global autoincrement" [*MobiLink - Server Administration*].

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.

You can also set this option using the ulinfo utility:

```
ulinfo -g ID ...
```

To set the global_database_id option from a client application, use the set database ID function for the programming interface you are using.

See also

- "Change UltraLite persistent database option settings" on page 167
- "UltraLite Information utility (ulinfo)" on page 196
- "SET OPTION statement [UltraLite] [UltraLiteJ]" on page 404
- "Using GLOBAL AUTOINCREMENT in UltraLite" on page 95
- UltraLite for C/C++: "SetDatabaseOptionInt method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "ULSetDatabaseID method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "DatabaseID property" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "setDatabaseID method" [*UltraLite - M-Business Anywhere Programming*]

Example

To autoincrement UltraLite database columns from 3001 to 4000, set the global ID to 3.

```
SET OPTION global_database_id=3;
```

UltraLite ml_remote_id option

A unique identifier in UltraLite that is used for MobiLink synchronization.

Allowed values

Any value that uniquely identifies the database for MobiLink synchronization.

Default

Null

Remarks

The **remote ID** is a unique identifier for an UltraLite remote that is used for MobiLink synchronization. The MobiLink remote ID is initially set to NULL. During the first synchronization, the MobiLink server sets the remote ID to a GUID. However, the remote ID can be any string that has meaning to you, if the string remains unique among all remote MobiLink clients. This uniqueness requirement is always enforced.

The remote ID stores the synchronization progress for the MobiLink user name. By including a unique remote ID, 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 client database. In this case, your synchronization scripts should reference the remote ID and not just the user name.

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.

See also

- "Change UltraLite persistent database option settings" on page 167
- "Remote IDs" [*MobiLink - Client Administration*]
- "Grant REMOTE permission" [*SQL Remote*]
- "UltraLite Information utility (ulinfo)" on page 196
- "SET OPTION statement [UltraLite] [UltraLiteJ]" on page 404
- UltraLite for C/C++: "SetDatabaseOption method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "ULSetDatabaseOptionString method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "SetDatabaseOption method" [*UltraLite - .NET Programming*]
- "UltraLite clients" on page 93
- "UltraLite user authentication" [*MobiLink - Client Administration*]
- "User Name synchronization parameter" on page 131
- and "Password synchronization parameter" on page 120

Change UltraLite persistent database option settings

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

To browse or modify persistent UltraLite database options (Sybase Central)

1. Connect to the database.
2. Right-click the database and choose **Options**.
3. If you want 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.

See also

- "SET OPTION statement [UltraLite] [UltraLiteJ]" on page 404
- "Accessing UltraLite database properties" on page 162
- "DB_PROPERTY function [System]" on page 299
- UltraLite C/C++: "GetDatabaseProperty method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "GetDatabaseProperty method" [*UltraLite - .NET Programming*]
- UltraLite for M-Business Anywhere: "getDatabaseProperty method" [*UltraLite - M-Business Anywhere Programming*]

UltraLite connection parameters

UltraLite CACHE_SIZE connection parameter

Defines the size of the database cache.

Syntax

CACHE_SIZE=*number*{ **k** | **m** | **g** }

Default

The default 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 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 even 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 - added to each index on the table. So, while synchronizing, the cache will tend to hold the pages where the new rows are stored, as well as the index pages for the current table. Synchronization performance will depend greatly 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.

If the cache size is not specified, or if you set the size to 0, the default size is used. If your testing shows the need for better performance, you should increase the cache size. For Windows, the default cache is set to about 1.5% of total physical memory, or 110% of the database file size, whichever is less. For Windows Mobile, the default cache size is limited to the lesser of: 25% of total physical memory, 90% of available physical memory less 1MB, or 110% of the database file size.

By default, the size is in bytes. Use k, m, or g to specify units of kilobytes, megabytes, or gigabytes, respectively.

If you exceed the maximum cache size, it is automatically replaced with your platform's upper cache size limit. Increasing the cache size beyond the size of the database does not provide any performance improvement, and a large cache size can interfere with the number of other applications you can use.

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.

See also

- “Opening UltraLite connections with connection strings” on page 37
- “UltraLite page_size creation parameter” on page 146
- “UltraLite RESERVE_SIZE connection parameter” on page 181
- UltraLite for C/C++: “Connecting to a database” [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: “OpenConnection method” [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: “Connecting to a database” [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: “Connecting to an UltraLite database” [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: “openConnection method” [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: “Connecting to a database” [*UltraLite - .NET Programming*]
- UltraLite.NET: “Open method” [*UltraLite - .NET Programming*]

Example

The following connection string fragment sets the cache size to 2 MB.

```
"CACHE_SIZE=2m"
```


UltraLite CE_FILE connection parameter

Names the new database file when creating a database. When opening a connection to an existing database, this connection parameter identifies the database.

Syntax

CE_FILE=*path\ce-db*

Default

DBF connection parameter.

Behavior

1. If DBF is specified, look for a matching database (identical filename) and connect if found, proceed to auto-start if not.

2. A database is auto-started when required if DBF is specified.

Remarks

You should use the CE_FILE connection parameter for UltraLite client applications that use the same connection string to connect to a Microsoft Windows Mobile device, and other platforms.

The CE_FILE connection parameter takes precedence over the DBF connection parameter. If you are connecting from an UltraLite administration tool, or your connection object only connects to a Windows Mobile database, use the DBF connection parameter.

The value of CE_FILE must meet the file name requirements for Windows Mobile. If you include an absolute path to the database, then all directories must exist before setting the path to this file. UltraLite does not create them automatically.

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.

See also

- “Opening UltraLite connections with connection strings” on page 37
- “Specifying file paths in an UltraLite connection parameter” on page 36
- “Precedence of connection parameters for UltraLite administration tools” on page 38
- “UltraLite DBF connection parameter” on page 172
- UltraLite for C/C++: “Connecting to a database” [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: “OpenConnection method” [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: “Connecting to a database” [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: “Connecting to an UltraLite database” [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: “openConnection method” [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: “Connecting to a database” [*UltraLite - .NET Programming*]
- UltraLite.NET: “Open method” [*UltraLite - .NET Programming*]

Example

The following example creates a new connection and identifies different database files for the Windows desktop and Windows Mobile platforms:

```
Set Connection = DatabaseMgr.OpenConnection("DBF=d:\Dbfile.udb;CE_FILE=\myapp
\MyDB.udb")
```

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.

See also

- "Flushing single or grouped transactions" on page 89
- "UltraLite commit_flush_count option [temporary]" on page 163
- "UltraLite commit_flush_timeout option [temporary]" on page 164
- "CHECKPOINT statement [UltraLite]" on page 375
- UltraLite embedded SQL: "ULCheckpoint method" [*UltraLite - C and C++ Programming*]
- UltraLite C++: "Checkpoint method" [*UltraLite - C and C++ Programming*]

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.

See also

- "Opening UltraLite connections with connection strings" on page 37
- "UltraLite DBN connection parameter" on page 174
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

Example

The following connection string fragment sets the first connection name to MyFirstCon.

```
"CON=MyFirstCon"
```

UltraLite DBF connection parameter

Names the new database file when creating a database. When opening a connection, this connection parameter indicates which database file you want to load and connect to.

Syntax

DBF=*ul-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

Because they are aliases, if DBF is used concurrently, the last one specified takes precedence.

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:

- CE_FILE
- desktop
- device
- NT_FILE

If specified, these platform-specific connection parameters take precedence over DBF.

The value of DBF must meet the file name requirements for the platform.

Windows Mobile If you are deploying to a Windows Mobile device, UltraLite utilities and wizards can administer an UltraLite database on an attached Windows Mobile device. To identify a file on a Windows Mobile device, you must specify the required absolute path, and use the **wce:** prefix.

Any leading or trailing spaces in parameter values are ignored. The value cannot include leading single quotes, leading double quotes, or semicolons.

See also

- "Opening UltraLite connections with connection strings" on page 37
- "Specifying file paths in an UltraLite connection parameter" on page 36
- "Precedence of connection parameters for UltraLite administration tools" on page 38
- "UltraLite DBN connection parameter" on page 174
- "UltraLite CE_FILE connection parameter" on page 169
- "UltraLite NT_FILE connection parameter" on page 179
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

Examples

To connect to the database, *MyULdb.udb*, installed in the desktop directory *c:\mydb*, use the following connection string:

```
"DBF=c:\mydb\MyULdb.udb"
```

To connect to the same database that is deployed to the *UltraLite* folder of the attached Windows Mobile device, use the following connection string:

```
"DBF=wce:\UltraLite\MyULdb.udb"
```

UltraLite DBKEY connection parameter

Provides an encryption key for the database when creating a new database. When opening a connection to an existing database, this connection parameter 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 using either the AES 256-bit or AES FIPS algorithm. By using strong encryption, you have increased security against skilled and determined attempts to gain access to the data. However, the use of strong encryption has a significant performance impact.

Any leading or trailing spaces in parameter values are ignored. The value cannot include leading single quotes, leading double quotes, or semicolons.

See also

- "Opening UltraLite connections with connection strings" on page 37
- "Securing UltraLite databases" on page 32
- "UltraLite obfuscate creation parameter" on page 146
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

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.

See also

- "Opening UltraLite connections with connection strings" on page 37
- "UltraLite DBF connection parameter" on page 172
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

Example

Use the following parameters to connect to the running UltraLite database named Kitchener:

```
DBN=Kitchener;DBF=cities.udb
```

UltraLite desktop connection parameter

Names the new database file when creating a database. When opening a connection to an existing database, this connection parameter identifies the database.

Syntax

desktop:DBF=*path\db* | **temp_dir**=*\Temp*

Remarks

You should use the desktop connection parameter for UltraLite client applications that use the same connection string to connect to a Microsoft Windows NT or Mac device.

Colon (**:**) is considered to be a separator in addition to underscore (**_**). Options with a prefix take precedence over options without a prefix.

A **temp_dir** connection parameter is now available. This must name a directory (which already exists). UL will place the temporary file (with name still derived from the database name) in the specified directory, rather than beside the database file (the default and previous behavior). 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.

The value of desktop must meet the file name requirements for Windows NT or Mac. If you include an absolute path to the database, then all directories must exist before setting the path to this file. UltraLite does not create them automatically.

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.

Behavior

1. If DBF is specified, look for a matching database (identical filename) and connect if found, proceed to auto-start if not.

2. A database is auto-started when required if DBF is specified.

See also

- "Opening UltraLite connections with connection strings" on page 37
- "Specifying file paths in an UltraLite connection parameter" on page 36
- "Precedence of connection parameters for UltraLite administration tools" on page 38
- "UltraLite DBF connection parameter" on page 172
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

Example

The following example creates a new connection and identifies different database files for the Windows desktop and Windows Mobile platforms:

```
"desktop:DBF=C:\dir\db.udb; device:DBF=\SD Card\db.udb; device:temp_dir=
\Temp; device:cache_size=4M"
```


UltraLite device connection parameter

Names the new database file when creating a database. When opening a connection to an existing database, this connection parameter identifies the database.

Syntax

device:DBF=*path\db* | **temp_dir=***\Temp*

Remarks

You should use the device connection parameter for UltraLite client applications that use the same connection string to connect to a Microsoft Windows CE or iPhone device.

Colon (**:**) is considered to be a separator in addition to underscore (**_**). Options with a prefix take precedence over options without a prefix.

A **temp_dir** connection parameter is now available. This must name a directory (which already exists). UL will place the temporary file (with name still derived from the database name) in the specified directory, rather than beside the database file (the default and previous behavior). 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.

The value of device must meet the file name requirements for Windows CE or iPhone. If you include an absolute path to the database, then all directories must exist before setting the path to this file. UltraLite does not create them automatically.

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.

See also

- "Opening UltraLite connections with connection strings" on page 37
- "Specifying file paths in an UltraLite connection parameter" on page 36
- "Precedence of connection parameters for UltraLite administration tools" on page 38
- "UltraLite DBF connection parameter" on page 172
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

Example

The following example creates a new connection and identifies different database files for the Windows desktop and Windows Mobile platforms:

```
"desktop:DBF=C:\dir\db.udb; device:DBF=\SD Card\db.udb; device:temp_dir=
\Temp; device:cache_size=4M"
```

UltraLite MIRROR_FILE connection parameter

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).

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, SQLE_MIRROR_FILE_MISMATCH is generated and the database will not start. When a corrupt page is recovered, the warning SQLE_CORRUPT_PAGE_READ_RETRY is generated. (Without mirroring, or if the mirror file is also corrupt, the error SQLE_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.) Specify the checksum_level database creation option to enable checksums. UltraLite will generate the warning SQLE_MIRROR_FILE_REQUIRES_CHECKSUMS if checksums are not enabled when using a mirror file. See “UltraLite checksum_level creation parameter” on page 136.

Note that because the mirror is an exact copy of the database file, it can be started directly as a database. The ulvalid utility will report corrupt pages. See “UltraLite Validate Database utility (ulvalid)” on page 218.

See also

- "Opening UltraLite connections with connection strings" on page 37
- "Specifying file paths in an UltraLite connection parameter" on page 36
- "Precedence of connection parameters for UltraLite administration tools" on page 38
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

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")
```

UltraLite NT_FILE connection parameter

Names the new database file when creating a database. When opening a connection to an existing database, the parameter identifies the database.

Syntax

NT_FILE=*path\nt-db*

Default

DBF connection parameter.

Remarks

You should use the NT_FILE connection parameter for UltraLite client applications that use the same connection string to connect to a desktop database, and a database on other platforms.

This connection parameter takes precedence over the DBF parameter. If you are connecting from an UltraLite administration tool, or your connection object only connects to a desktop database, use the DBF connection parameter.

The value of NT_FILE must meet the file name requirements for Windows desktop platforms.

The path can be absolute or relative. If you include a directory as part of the file name, then all directories must exist before setting the path to this file. UltraLite does not create them automatically.

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.

See also

- "Opening UltraLite connections with connection strings" on page 37
- "Specifying file paths in an UltraLite connection parameter" on page 36
- "Precedence of connection parameters for UltraLite administration tools" on page 38
- "UltraLite DBF connection parameter" on page 172
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

Example

The following example creates a new connection and identifies different database files for the desktop and Windows Mobile platforms:

```
Connection = DatabaseMgr.OpenConnection("UID=JDoe;PWD=ULdb;
NT_FILE=c:\test\MyTestDB.udb;CE_FILE=\database\MyCEDB.udb")
```

UltraLite PWD connection parameter

Defines the password for a user ID that is used for authentication.

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 Sybase Central. See "Managing user permissions and authorities" [*SQL Anywhere Server - Database Administration*].

See also

- "Opening UltraLite connections with connection strings" on page 37
- "UltraLite user authentication" on page 39
- "Interpreting user ID and password combinations" on page 40
- "UltraLite UID connection parameter" on page 184
- UltraLite for C/C++: "Authenticating users" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Authenticating users" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Authenticating users" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Authenticating users" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

Examples

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=''"
```

UltraLite RESERVE_SIZE connection parameter

Pre-allocates the file system space required for your UltraLite database, without actually inserting any data. By reserving the file system space means that the space cannot be 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).

See also

- "Opening UltraLite connections with connection strings" on page 37
- "UltraLite CACHE_SIZE connection parameter" on page 167
- "UltraLite page_size creation parameter" on page 146
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

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"
```

UltraLite START connection parameter

Starts the UltraLite engine executable. This parameter is only required if the engine is not in one of the expected locations. See "Starting the UltraLite engine" on page 45 for a list of the expected locations.

Syntax

START=*path***uleng12.exe**

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.

See also

- "Starting the UltraLite engine" on page 45
- "UltraLite Engine utility (uleng12)" on page 194
- "Choosing an UltraLite data management component" on page 19
- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

Example

The following command starts the UltraLite engine that is located in the *Program Files* directory:

```
Start=\"Program Files\uleng12.exe"
```

An alternative way to define this path is to put the entire string in single quotes:

```
Start='"\Program Files\uleng12.exe"'
```

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

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.

See also

- UltraLite for C/C++: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: "OpenConnection method" [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: "Connecting to a database" [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: "Connecting to an UltraLite database" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: "openConnection method" [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: "Connecting to a database" [*UltraLite - .NET Programming*]
- UltraLite.NET: "Open method" [*UltraLite - .NET Programming*]

Example

The following connection string fragment puts the temp file in the *\Temp* directory:

```
temp_dir=\Temp;
```

UltraLite UID connection parameter

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, 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.

See also

- “Opening UltraLite connections with connection strings” on page 37
- “UltraLite user authentication” on page 39
- “Interpreting user ID and password combinations” on page 40
- UltraLite for C/C++: “Authenticating users” [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: “Connecting to a database” [*UltraLite - C and C++ Programming*]
- UltraLite for C/C++: “OpenConnection method” [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: “Authenticating users” [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: “Connecting to a database” [*UltraLite - C and C++ Programming*]
- UltraLite for M-Business Anywhere: “Authenticating users” [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: “Connecting to an UltraLite database” [*UltraLite - M-Business Anywhere Programming*]
- UltraLite for M-Business Anywhere: “openConnection method” [*UltraLite - M-Business Anywhere Programming*]
- UltraLite.NET: “Authenticating users” [*UltraLite - .NET Programming*]
- UltraLite.NET: “Connecting to a database” [*UltraLite - .NET Programming*]
- UltraLite.NET: “Open method” [*UltraLite - .NET Programming*]

Example

The following connection string fragment supplies the user ID DBA and password sql for a database:

```
"UID=DBA;PWD=sql"
```

UltraLite utilities

UltraLite includes a set of utilities that are designed to perform basic database administration activities from 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.

Note
Options for the utilities documented in this section are case sensitive, unless otherwise noted. Type options *exactly* as they are displayed.

Supported exit codes

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.

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.

Interactive SQL for UltraLite utility (dbisql)

Executes SQL commands and runs command files against a database.

Syntax

dbisql -c "*connection-string*" [*options*] [*dbisql-command* | *command-file*]

dbisql -c "*connection-string*" **-ul** [*options*] [*dbisql-command* | *command-file*]

dbisql-command: A SQL statement or a series of sql statements separated by a command-delimiter.

Option	Description
@*data*	Reads in options from the specified environment variable or configuration file. If both the environment variable and configuration file exist with the same name, the environment variable is used. See "Using configuration files" [*SQL Anywhere Server - Database Administration*]. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" [*SQL Anywhere Server - Database Administration*].
-c *"keyword=value; ..."*	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. See "Connection parameters" [*SQL Anywhere Server - Database Administration*].
-d *delimiter*	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. This option overrides the setting of the command_delimiter option. See "command_delimiter option [Interactive SQL]" [*SQL Anywhere Server - Database Administration*].
-d1	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.
-datasource *DSN-name*	Specifies an ODBC data source to connect to.

Option	Description
-f *filename*	Opens (but does not run) in the SQL Statements pane the file called *filename*. If the -f option is given, the -c option is ignored; that is, no connection is made to the database. The file name can be enclosed in quotation marks, and *must* be enclosed in quotation marks if the file name contains a space. 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. If the file name does not include a full drive and path specification, it is assumed to be relative to the current directory. This option is only supported when Interactive SQL is run as a windowed application.
-host *hostname*	Specifies the *hostname* or IP address of the computer on which the database server is running. You can use the name localhost to represent the current computer.
-nogui	Runs Interactive SQL as a console application, with no windowed user interface. This is useful for batch operations. If you specify either *dbisql-command* or *command-file*, then -nogui is assumed. In this 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. See “Software component exit codes” [*SQL Anywhere Server - Programming*].

Option	Description
-onerror { **continue** | **exit** }	Controls what happens if an error is encountered while reading statements from a command file. It is useful when using Interactive SQL in batch operations. This option overrides the on_error setting. See "on_error option [Interactive SQL]" [*SQL Anywhere Server - Database Administration*]. Define one of the following supported *behavior* values: • **Continue** The error is ignored and Interactive SQL continues executing statements. • **Exit** Interactive SQL terminates.
-q	Suppresses output messages. Sets the utility to run in quiet mode. This is useful only if you start Interactive SQL with a command or command file. Specifying this option does not suppress error messages, but it does suppress the following: • warnings and other non-fatal messages • the printing of result sets
-ul	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. By default, Interactive SQL assumes that you are connecting to SQL Anywhere databases. When you specify the -ul 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 **Change Database Type** dropdown list on the **Connect** window. For more information about connecting to UltraLite databases from Interactive SQL, see "Interactive SQL for UltraLite utility (dbisql)" on page 186.

Option	Description
-version	Displays the version number of Interactive SQL. You can also view the version number from within Interactive SQL; from the **Help** menu, choose **About Interactive SQL**.
-x	Scans commands but does not execute them. This is useful for checking long command files for syntax errors. For detailed descriptions of SQL statements and Interactive SQL commands, see "SQL language elements" [*SQL Anywhere Server - SQL Reference*].
dbisql-command | *command-file*	Execute the SQL statement or execute the specified *command-file*. If you do not specify a *SQL-statement* or *command-file*, Interactive SQL enters interactive mode, where you can type a command into a command window.

Remarks

Interactive SQL allows you to browse the database, execute SQL commands, and run command files. It also provides feedback about:

- the number of rows affected
- the time required for each command
- the execution plan of queries
- any error messages

You can connect to both SQL Anywhere and UltraLite databases. For information about connecting to SQL Anywhere databases, see "Interactive SQL utility (dbisql)" [*SQL Anywhere Server - Database Administration*].

Interactive SQL is supported on Windows, Solaris, Linux, and Mac OS X. See http://www.sybase.com/detail?id=1061806.

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 default_isql_encoding option. See "default_isql_encoding option [Interactive SQL]" [*SQL Anywhere Server - Database Administration*].

You can specify the encoding to use when reading or writing files using the ENCODING clause of the INPUT, OUTPUT, or READ statement. See:

- "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "OUTPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "READ statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

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). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

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, 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 Sybase Central, choosing **File** » **Open Interactive SQL**.
- From the **Start** menu by choosing **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Interactive SQL**.
- Using the dbisql command from a command prompt.

See also

- "Interactive SQL SQL statements" [*SQL Anywhere Server - Database Administration*]
- "Using Interactive SQL" [*SQL Anywhere Server - Database Administration*]
- "Using configuration files" [*SQL Anywhere Server - Database Administration*]
- "File Hiding utility (dbfhide)" [*SQL Anywhere Server - Database Administration*]
- "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "OUTPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "READ statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "UltraLite connection parameters" on page 167
- "Supported exit codes" on page 185

Example

The following command runs the command 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 command file, the process terminates.

```
dbisql -ul -c DBF=CustDB.udb -onerror exit mycom.sql
```

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.
-e *flag*	This option flags as an error any static embedded SQL that is not part of a specified standard. The *level* value indicates the standard to use. For example, `sqlpp -e c03 ...` flags any syntax that is not part of the core SQL/2003 standard. The allowed values of *level* are: • **c03** Flag syntax that is not core SQL/2003 syntax • **p03** Flag syntax that is not full SQL/2003 syntax • **c99** Flag syntax that is not core SQL/1999 syntax • **p99** Flag syntax that is not full SQL/1999 syntax • **e92** Flag syntax that is not entry-level SQL/1992 syntax • **i92** Flag syntax that is not intermediate-level SQL/1992 syntax • **f92** Flag syntax that is not full-SQL/1992 syntax • **t** Flag non-standard host variable types • **u** Flag syntax that is not supported by UltraLite For compatibility with previous SQL Anywhere versions, you can also specify e, I, and f, which correspond to e92, i92, and f92, respectively.
-h *width*	Limits the maximum length of split lines output by sqlpp to *width* in the *.c* file. Back-slash 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).
-k	Notify the preprocessor that the program to be compiled includes a user declaration of SQLCODE.

Option	Description
-n	Generate line number information in the C file by using `#line` directives in the appropriate places in the generated code. Use this option to the report source errors and to debug source on line numbers in the *esql-filename* file, rather than in the *output-filename* file.
-o *O/S spec*	Not applicable to UltraLite.
-q	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
-r	Not applicable to UltraLite.
-s *string-length*	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.
-u	Required for UltraLite. Generate output specifically required for UltraLite databases.
-w *level*	Flag non-conforming SQL syntax as a warning. The *level* value indicates the standard to use. For example, `sqlpp -w c03 ...` flags any SQL syntax that is not part of the core SQL/2003 syntax. The allowed values of *level* are: • **c03** Flag syntax that is not core SQL/2003 syntax • **p03** Flag syntax that is not full SQL/2003 syntax • **c99** Flag syntax that is not core SQL/1999 syntax • **p99** Flag syntax that is not full SQL/1999 syntax • **e92** Flag syntax that is not entry-level SQL/1992 syntax • **i92** Flag syntax that is not intermediate-level SQL/1992 syntax • **f92** Flag syntax that is not full-SQL/1992 syntax • **t** Flag non-standard host variable types • **u** Flag syntax that is not supported by UltraLite For compatibility with previous SQL Anywhere versions, you can also specify e, I, and f, which correspond to e92, i92, and f92, respectively.

Option	Description
-x	Change multibyte strings to escape sequences, so that they can be passed through a compiler.
-z *collation-sequence*	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 *sqc*. 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 the command **ulinit -Z**.

Tip
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. For an overview of the SQL Flagger, see "Testing SQL compliance using the SQL Flagger" [*SQL Anywhere Server - SQL Usage*].

See also

- "Embedded SQL" [*SQL Anywhere Server - Programming*]
- "UltraLite character sets" on page 30

Example

The following command preprocesses the *srcfile.sqc* embedded SQL file in quiet mode for an UltraLite application.

```
sqlpp -u -q MyEsqlFile.sqc
```

UltraLite Engine utility (uleng12)

Manages concurrent UltraLite database connections from applications on 32- and 64-bit Windows Desktop, 32-bit Linux and Windows Mobile.

Syntax

uleng12

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.

See:

- "Deploying UltraLite to devices" on page 41
- "Compiling and linking your application" [*UltraLite - C and C++ Programming*]
- "Deploy UltraLite with AES_FIPS database encryption" on page 46
- "Deploy UltraLite with TLS-enabled synchronization" on page 47

See also

- "Working with UltraLite databases" on page 52
- "Choosing an UltraLite data management component" on page 19
- "UltraLite Engine Stop utility (ulstop)" on page 195
- "UltraLite START connection parameter" on page 182

UltraLite Engine Stop utility (ulstop)

Stops the UltraLite engine on 32-bit Windows desktops and Windows Mobile.

Syntax

ulstop

Remarks

Use ulstop during development to shut down the engine manually. You typically do not require ulstop in live deployments.

See also

- "Choosing an UltraLite data management component" on page 19
- "UltraLite Engine utility (uleng12)" on page 194

UltraLite Erase database (ulerase)

Erases an UltraLite database.

Syntax

ulerase[*options*] [*db-file-name*]

Option	Description
-k *key* OR **--ek**=*key*	Specify the encryption key for an encrypted database.
-p OR **--ep**	Specify that you want to be prompted for the encryption key.
--log	Log operations to the specified file.
-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.
-u <*uid*>,<*pwd*> OR **--dba**=<*uid*>,<*pwd*>	Specify the userid and password required to access the database.
-?	Displays utility usage information and exits.
dbname	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**.

UltraLite Information utility (ulinfo)

Displays information about an UltraLite database.

Syntax

ulinfo -c *options*

Option	Description
-c "*keyword=value;...*" OR **--connect**"*keyword=value;...*"	Supply database connection parameters. Required.
-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.
--log=*filename*	Log operations to the specified file.
-? OR **--help**	Displays utility usage information and exits.

Remarks

Warning messages generated when opening an UltraLite database are always displayed unless you use the -q option.

See also

- "UltraLite connection parameters" on page 167
- "UltraLite global_database_id option" on page 165
- "UltraLite ml_remote_id option" on page 166

Example

Show basic database internals for a file named *cv_dbattr.udb* that has already been synchronized:

```
ulinfo -c DBF=cv_dbattr.udb
```

Show database internals for a file named *CustDB.udb* and display database properties by enabling verbose messaging:

```
ulinfo -c DBF=CustDB.udb -v
```

UltraLite Initialize Database utility (ulinit)

Creates a new UltraLite database. This utility will either create an empty database with characteristics specified with the command line arguments ("empty mode"), or it can create a database based on a SQL Anywhere database "extract mode". In the latter case, an initial schema is created that matches tables and indexes in the SQL Anywhere reference database. Moreover, many of the SQL Anywhere reference database characteristics will be used for the new UltraLite database.

In the usage below, some options are only permissible in "extract mode," while others are only permissible in "empty mode." If no mode is stated in the description, then the option can be used in either mode.

Syntax

ulinit *options dbname*

Option	Description
-a " *keyword=value;...*" OR **--SAconnect**="*keyword=value;...*"	Sets the utility to "extract" mode and connects to an existing database using the specified connection parameters (extract mode). If this option is not present, the utility creates a new database using the specified connection parameters (empty mode).
-c OR **--case**	Empty mode. Enforce case sensitivity on all string comparisons.
-d OR **--datacopy**	Extract mode. For each table in the new UltraLite database, copy data from the corresponding table in the SQL Anywhere database. By default, this data will not be uploaded in subsequent synchronizations. To include the data in the next upload synchronization, use **-i** with **-d**.
--date_format=*format*	Empty mode. Sets the format for dates retrieved from the database. See "UltraLite date_format creation parameter" on page 138
--date_order=*date-format-interpretation*	Empty mode. Sets the interpretation of the date format. See "UltraLite date_order creation parameter" on page 141.

Option	Description
-e *value* OR **--fips**=*value*	Empty mode. On or off, 1 or 0, etc. Controls AES FIPS compliant data encryption, by using a Certicom certified cryptographic algorithm. FIPS encoding is a form of strong encryption. See "Securing UltraLite databases" on page 32 and "UltraLite fips creation parameter" on page 142.
-f OR **---exactschema**	Extract mode. Fail if exact schema is not supported in UltraLite; otherwise, warnings will appear if schema differs.
-g *id* OR **--databaseid**=*id*	Set the initial 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. See "UltraLite global_database_id option" on page 165.
-i OR **--insertforupload**	Extract mode. Use with **-d**. Include inserted rows in the next upload synchronization. By default, rows inserted by this utility are not uploaded during synchronization.
-k *key* OR **--key**=*key*	Extract mode. Specify the encryption key for an encrypted database.
-K OR **--prompt**	Empty mode. Specify that you want to be prompted for the encryption key.
-l *logfile* OR **--sql**=*logfile*	Extract mode. Log DDL database schema creation SQL statements, as executed, to *logfile*.

Option	Description
--log=*filename*	Empty mode. Log operations to the specified file.
-m *filename* OR **--mirror_file**=*filename*	Extract mode. Specify the database mirror file. See "UltraLite MIRROR_FILE connection parameter" on page 178.
--max_hash_size=*size*	Empty mode. Sets the default index hash size in bytes. See "UltraLite max_hash_size creation parameter" on page 143.
-n *pubname* OR **--publication**=*pubname*	Extract mode. Required. Add tables to the UltraLite database schema. *pubname* specifies a publication in the reference database. Tables in the publication are added to the UltraLite database. 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 **-n***.
--nearest_century=*yy*	Empty mode. Controls the interpretation of two-digit years in string-to-date conversions. See "UltraLite nearest_century creation parameter" on page 144.
-o *value* OR **--obfuscate**=*value*	Empty mode. On or off, 1 or 0, etc. Controls whether data in the database is obfuscated. Obfuscation is a form of simple encryption. See "Securing UltraLite databases" on page 32 and "UltraLite obfuscate creation parameter" on page 146.

Option	Description
-p *size* OR **--page_size**=*size*	Empty mode. Specify the database page size.
--precision=*precision*	Empty mode. Specifies the maximum number of digits in decimal point arithmetic results. See "UltraLite precision creation parameter" on page 148.
-q OR **--quiet**	Empty mode and extract mode. Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages and warnings are still displayed, however.
-r *size* OR --reserve_size	Database connection only. Reserve size. See "UltraLite RESERVE_SIZE connection parameter" on page 181.
-s *pubname* OR **--sync_publication**	Extract mode. Create a publication in the UltraLite database with the same definition as *pubname* in the reference database. Publications are used to configure synchronization. Supply more than one -s option to name more than one synchronization publication. Note that the tables in this publication must be included in a publication listed by the -n option. If -s is not supplied, the UltraLite remote has no named publications. For more information about how to create publications for MobiLink synchronization, see "Publications in UltraLite" on page 102.

Option	Description
-S *checksum_level* OR **--checksum_level**=*checksum_level*	Empty mode. 0, 1, or 2. Specifies the checksum level validation on database pages. See "UltraLite checksum_level creation parameter" on page 136.
--scale=*scale*	Empty mode. Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision. See "UltraLite scale creation parameter" on page 149.
-t *file* OR **--rootcert**=*file*	Empty mode and extract mode. Specify the file containing the trusted root certificate. This certificate is required for server authentication.
--time_format=*format*	Empty mode. Sets the format for times retrieved from the database. See "UltraLite time_format creation parameter" on page 150.
--timestamp_format=*format*	Empty mode. Sets the format for timestamps retrieved from the database. See "UltraLite timestamp_format creation parameter" on page 152.
--timestamp_increment=*increment*	Empty mode. Determines how the timestamp is truncated in UltraLite. See "UltraLite timestamp_increment creation parameter" on page 154.
--timestamp_with_time_zone_format= *format*	Empty mode. This option sets the format for TIMESTAMP WITH TIME ZONE values retrieved from the database. See ."UltraLite timestamp_with_time_zone_format creation parameter" on page 155

Option	Description
-u *<uid>,<pwd>* OR **--dba**=*<uid>,<pwd>*	Database connection only. Specify the userid and password.
--utf8_encoding=*value*	Empty mode. On or off, 1 or 0, etc. Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode. See "UltraLite character sets" on page 30 and "UltraLite utf8_encoding creation parameter" on page 157.
-w OR **--nowarnings**	Extract mode. Do not display warnings.
-x *table* OR **--exclude**	Extract mode. Exclude the tables named in the list.
-y OR **--overwrite**	Empty mode and extract mode. Over-write the existing database file.
-z *collation-sequence* OR **--collation**=*collation-sequence*	Empty mode. Specify the collation sequence.
-Z OR **--listcollation**	Empty mode. List the available collation sequences and exit.

Option	Description
-? OR **--help**	Display utility usage and exit.

Remarks

The SQL Anywhere reference database, when used, acts as the source for:

- database configuration (for example, the collation sequence used)
- table definitions
- synchronization publications

Together they help create the UltraLite schema—information that defines the structure of the new UltraLite database. However, the new database you create is initially empty unless you use the **-d** option.

When run in extract mode, ULINIT will attempt 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 will be ignored and the UltraLite default used instead. A warning will be generated and creation will continue. 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. If the schema will not support a reasonable UltraLite alternative, ulinit will fail.

If you want to initialize an UltraLite database from an RDBMS other than SQL Anywhere, use the **Create Synchronization Model Wizard** in Sybase Central. When you run the wizard, you are prompted to connect to a consolidated database to obtain schema information.

UltraLite uses the name of the collation sequence that was defined in the reference database. However, you can still choose to use UTF-8 to encode the database, by setting the utf8_encoding property.

To see a list of supported collations (and corresponding codepages), run **ulinit -Z** at a command prompt. If your collation sequence is not supported by UltraLite, you should change it to one that is. For example, if your reference database collation is the UCA collation, you should:

1. Unload the reference database and then reload it with a different collation.

2. Run ulinit on this new version of the database.

See also

- “Create an UltraLite database from a SQL Anywhere reference database” on page 25
- “Introduction to synchronization models” [*MobiLink - Getting Started*]
- “UltraLite connection parameters” on page 167

Examples

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
```

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
-a OR **--append**	Add data and schema definitions into an existing database.
-c "*keyword=value;...*" OR **--connect** "*keyword=value;...*"	Supply the database connection parameters.

Option	Description
-d OR **--dataonly**	Load data only, ignoring any schema meta-data 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 **--fips**= *value*	On or off, 1 or 0, etc. Controls AES FIPS compliant data encryption, by using a Certicom certified cryptographic algorithm. FIPs encoding is a form of strong encryption. See "Securing UltraLite databases" on page 32 and "UltraLite fips creation parameter" on page 142.
-E *behavior* OR **--onerror**=*behavior*	Control what happens if an error is encountered while reading data from the XML file. Specify one of the following supported *behavior* values: • **continue** ulload ignores the error and continues to load XML. • **prompt** ulload prompts you to see if you want to continue. • **quit** ulload stops loading the XML and terminates with an error. This behavior is the default behavior if none is specified. • **exit** ulload exits.
-f *directory* OR **--filedir**=*directory*	Set the directory that contains files with additional data to load. See "UltraLite Database Unload utility (ulunload)" on page 214.
-g *ID* OR **--databaseid**=*ID*	Set the initial 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. See "UltraLite global_database_id option" on page 165.
-i OR **--insertforsync**	Include inserted rows in the next upload synchronization. By default, rows inserted by this utility are not uploaded during synchronization.

Option	Description
-l *filename* OR **--log**=*filename*	Log operations to the specified file.
-n OR **--schemaonly**	Load schema meta-data only, ignoring any data in the XML input file.
-o *value* OR **--obfuscate**=*value*	On or off, 1 or 0, etc. Controls whether data in the database is obfuscated. Obfuscation is a form of simple encryption. See "Securing UltraLite databases" on page 32 and "UltraLite obfuscate creation parameter" on page 146.
-p *page-size* OR **--page_size**=*page-size*	Defines the database page size. See "UltraLite page_size creation parameter" on page 146.
-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 *file* OR **--sql**=*file*	Log the SQL statements used to load the database into the specified *file*.
-t *file* OR **--rootcert**=*file*	Specify the file containing the trusted root certificate. This certificate is required for server authentication.
--utf8_encoding=*value*	On or off, 1 or 0, etc. Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode. See "UltraLite character sets" on page 30 and "UltraLite utf8_encoding creation parameter" on page 157.

Option	Description
-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 meta-data for the schema and/or meta-data for the database data. **-d** ignores the schema meta-data, only adding data to the *.udb* file. **-n** ignores the data and the meta-data, 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.

See also

- "UltraLite connection parameters" on page 167
- "UltraLite Database Unload utility (ulunload)" on page 214
- "Supported exit codes" on page 185
- "UltraLite global_database_id option" on page 165

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
```

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 -c [*options*] [*synchronization parameters*]

Option	Description
-c "*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.
--log	Log operations to the specified file.

-p *profile-name*
 	
OR
 	
--profile=profile
	Synchronize using the named sync profile, equivalent to:
	
	`synchronize profileName merge 'syncOptions'`
	
	where sync options are taken from the trailing ulsync options. For example:
	
	`ulsync -p profileName "MobiLinkUid=ml;ScriptVersion=Version001...syncOptions"`
	
	See “Synchronization profile options” on page 212.

-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 **--verbose**	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. See "CREATE SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 379.

Remarks

When a certificate file is specified either with the **trusted_certificates** or the **e2ee_public_key** option, the UltraLite runtime will look for these files only in the main resource bundle, which is a part of every iPhone application deployment package. Add items to this bundle by including them in the */Resources* folder in your xCode project. This is not applicable to certificates that are stored in the Ultralite database, and does not affect Mac OS clients (only iPhone). See "trusted_certificates" [*MobiLink - Client Administration*]and "e2ee_type" [*MobiLink - Client Administration*].

The following options that were valid for versions 10 and preceding 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** have been folded into the **Stream=<stream{<stream-parms>}** sync parameters string.

ulsync can be considered to be equivalent to one the following SQL statements, depending on usage:

```
ulsync -p <profile> "<parms>"
```

is equivalent to:

```
 SYNCHRONIZE PROFILE <profile> MERGE
<parms>
```

and

```
ulsync "<parms>"
```

is equivalent to:

```
 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_certificates=***file* stream option.

This utility returns error codes. Any value other than 0 means that the operation failed.

See also

- “Synchronization profile options” on page 212
- “End-to-end encryption” [*SQL Anywhere Server - Database Administration*]
- “trusted_certificates” [*MobiLink - Client Administration*]
- “UltraLite connection parameters” on page 167
- “UltraLite clients” on page 93
- “Supported exit codes” on page 185
- “MobiLink File Transfer utility (mlfiletransfer)” [*MobiLink - Client Administration*]
- “Deploy UltraLite with TLS-enabled synchronization” on page 47

Examples

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:\certs\rsa.crt* certificate. The **trusted_certificates=***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_certificates=c:\certs
\rsa.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 Database Synchronize Utility Version XX.X
    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:\Documents and Settings\All Users\Documents\SQL Anywhere
12\Samples\UltraLite\SyncEncrypt\custdb.udb"
MobiLinkUid=50;ScriptVersion=custdb 12.0;Stream=tcpip{port=2439}
```

The following command illustrates how to use TLS encryption with E2EE:

```
ulsync -c "uid=dba;pwd=sql;dbf=myudb.db"
"MobiLinkUid=rem1;MobiLinkPwd=password;ScriptVersion=v1;Stream=tls{host=mySer
ver;port=2439;trusted_certificates=c:
\clientcert.pem;e2ee_type=rsa;e2ee_public_key=c:\e2eepublic.pem}"
```

Synchronization profile options

You specify synchronization profile options with the ulsync utility on the command line after you have defined all other command line options you want to use. The keywords are case insensitive.

Synchronization profile option	Valid values	Description
AllowDownloadDupRows	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 synchronisation to fail. The default value is "no." See "Additional Parameters synchronization parameter" on page 111
AuthParms	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. See "Authentication Parameters synchronization parameter" on page 112.
CheckpointStore	Boolean	Adds additional checkpoints of the database during synchronization to limit database growth during the synchronization process. See "Additional Parameters synchronization parameter" on page 111.
ContinueDownload	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. See "Resuming failed downloads" [*MobiLink - Server Administration*].
DisableConcurrency	Boolean	Disallow database access from other threads during synchronization. See "Additional Parameters synchronization parameter" on page 111.
DownloadOnly	Boolean	Performs a download-only synchronization. See "Download Only synchronization parameter" on page 115.
KeepPartialDownload	Boolean	Controls whether UltraLite keeps a partial download if a communication error occurs. By default, UltraLite does not roll back partially downloaded changes. See "Keep Partial Download synchronization parameter" on page 117.
MobiLinkPwd	String	Specifies the existing MobiLink password associated with the user name. See "MobiLinkPwd (mp) extended option" [*MobiLink - Client Administration*].
MobiLinkUid	String	Specifies the MobiLink user name. See "-u dbmlsync option (deprecated)" [*MobiLink - Client Administration*]. See "-mn dbmlsync option" [*MobiLink - Client Administration*].

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. See "-mn dbmlsync option" [*MobiLink - Client Administration*].
Ping	Boolean	Confirms communications with the server only; no synchronization is performed. See "Ping synchronization parameter" on page 121.
Publications	String (comma separated)	Specifies the publications(s) to synchronize. The publications determine the tables on the remote 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. See "Publications in UltraLite" on page 102.
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. See "ScriptVersion (sv) extended option" [*MobiLink - Client Administration*].
SendColumnNames	String	Specifies that column names should be sent to the MobiLink server as part of the upload file when synchronizing. By default, column names are not sent. See "Send Column Names synchronization parameter" on page 124.
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. See "Send Download Acknowledgement synchronization parameter" on page 125.
Stream	String (with sublist)	Specifies the MobiLink network synchronization protocol. See "Stream Type synchronization parameter" on page 127.
TableOrder	String (comma separated)	Specifies the order of tables in the upload. By default, UltraLite selects an order based on foreign key relationships. See "Additional Parameters synchronization parameter" on page 111.
UploadOnly	String	Specifies that synchronization will only include an upload, and no download will occur. See "Upload Only synchronization parameter" on page 130.

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.

See also

- “ALTER SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]” on page 369
- “DROP SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]” on page 393
- “SYNCHRONIZE statement [UltraLite] [UltraLiteJ]” on page 407
- “UltraLite creation parameters” on page 135

UltraLite Database Unload utility (ulunload)

Unloads any of the following, depending on the options used:

- An entire UltraLite database to XML or SQL.
- All or part of UltraLite data only to XML or SQL.

Syntax

ulunload -c "*connection-string*" [*options*] *output-file*

Option	Description
-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 "*keyword=value;...*" OR **--connect**="*keyword=value;...*"	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: `-e mydbtable1,mydbtable5`
-f *directory* OR **--filedir**=*directory*	Set the directory to store data larger than the maximum size specified by -b. The default is the same directory as the output file.

Option	Description
-l *filename* OR **--log**=*filename*	Unload schema only, ignoring any data in the database.
-n OR **--schemaonly**	Unload schema only, ignoring any data in the database.
-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 OR **--sql**	Unload as SQL Anywhere-compatible SQL statements. SQL file output can be read by UltraLite or SQL Anywhere using DBISQL.
-t *table,...* OR **--include**=*table,...*	Unload data in the named *table* only. You can name multiple tables in a comma separated list. For example: `-t mydbtable2,mydbtable6`
-v OR **--verbose**	Print verbose messages.
-x *owner* OR **--owner**=*owner*	Output tables so they are owned by a specific user ID. You can use this option with the -s option.
-y OR **--overwrite**	Overwrite *output-file* without confirmation.

Option	Description
output-file	Required. Set the name of the file that the database is unloaded into. If you use the -s 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 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.

See also

- "UltraLite connection parameters" on page 167
- "Supported exit codes" on page 185
- "UltraLite Load XML to Database utility (ulload)" on page 205
- "UltraLite Information utility (ulinfo)" on page 196

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 *sample1.sql*. Overwrite the SQL file if it exists.

```
ulunload -c DBF=sample.udb -d -y sample.sql
```

UltraLite Unload Old Database utility (ulunloadold)

Unloads UltraLite version 8.0.2 to 9.0.x databases and/or schema files (*.*usm*) into an XML file.

Syntax

ulunloadold -c "*connection-string*" [*options*] *xml-file*

Option	Description
-b *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*"	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.
-f *directory*	Set the directory to store data larger than the maximum size specified by -b. The default is the same directory as the XML file.
-q	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
-v	Print verbose messages.
-y	Overwrite *xml-file* without confirmation.
xml-file	Set the name of the XML file that data will be unloaded into.

Remarks

UltraLite version 12 cannot directly upgrade version 8.x or version 9.x databases. Use this tool to generate an XML file that can then be used by ulload to create a version 12 database. Do not unload UltraLite version 12 databases with this utility. Use the ulunload utility instead.

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 name follows this naming convention:

```
tablename-columnname-rownumber.bin
```

Any error code value other than 0 means that the operation failed.

This utility cannot be used to unload databases directly on Windows Mobile devices. You must first copy them to a desktop computer.

See also

- "UltraLite connection parameters" on page 167
- "UltraLite Load XML to Database utility (ulload)" on page 205
- "UltraLite Database Unload utility (ulunload)" on page 214
- "UltraLite Information utility (ulinfo)" on page 196

Example

Upgrading an UltraLite 8.0.x schema file named *dbschema8.usm* into an UltraLite version 12 database named *db.udb* requires these two commands:

```
ulunloadold -c SCHEMA_FILE=dbschema8.usm dbschema.xml

ulload -c DBF=db.udb dbschema.xml
```

UltraLite Validate Database utility (ulvalid)

Performs a full ("normal") validation of an UltraLite database.

Syntax

ulvalid -c "*connection-string*" [*options*]

Option	Description
-c "*connection-string*"	Required. Connect to the database as identified in *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.

Option	Description
-e	Express validation. Only perform table validation. This option provides a faster validation than normal validation.
-q	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
-v	Print verbose messages.
-?	Displays utility usage information and exits.

Remarks

Validating a database verifies the accuracy of the table meta-data and ensures the file has not been corrupted.

The validation includes:

- **Database pages** Validate all database pages, using checksums when enabled. Note that certain critical pages always have checksums and even pages without checksums undergo a basic validity check. See "UltraLite checksum_level creation parameter" on page 136.
- **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.

See also

- "Validate an UltraLite database" on page 13

Example

An example of an express validation of a database named *sample.udb* run in quiet mode.

```
ulvalid -c DBF=sample.udb -e -q
```

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 can still be queried for backwards compatibility (they are in essence system views), but they only contain information about user schema (like tables, columns, indexes) not system schema. For example, you cannot query systable to find the properties of systable itself. You can only query 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. It is recommended that you use these objects and APIs to explore schema rather than querying the system views.

Note that all queries performed on these system views are equivalent to full table scans. Index scans are not supported on these system views.

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.
map_handle	UNSIGNED INT	Internal use only.
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 MobiLink synchronization. Can be one of either **no_sync** for no synchronization, **all_sync** to synchronize every row, or **normal_sync** for synchronize changed rows only.
table_type	VARCHAR(32)	**user** to indicate user-created tables.
tpd_handle	UNSIGNED INT	Internal user only.

Constraints

PRIMARY KEY (object_id)

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.

Column name	Column type	Description
domain_info	UNSIGNED INT	Used with a variable sized domain.
nulls	CHAR(1)	Determines if the column allows nulls default.
object_id	UNSIGNED INT	A unique identifier for that column.
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)

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 **foreign**.
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 **foreign**. 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 **foreign**. Lists the identifier of the primary index.
primary_table_id	UNSIGNED INT	Only required if type is **foreign**. 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.

Column name	Column type	Description
type	VARCHAR(10)	The type of index. Can be one of: • **primary** • **foreign** • **key** • **unique** • **index**
hash_size	UNSIGNED SHORT	Stores the hash size used for index hashing.

Constraints

PRIMARY KEY (table_id, object_id)

FOREIGN KEY(table_id) REFERENCES systable(object_id)

See also

- "sysixcol system table" on page 222

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	CHAR(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)

See also

- "sysindex system table" on page 221

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	CHAR(128)	The name of the publication.

Constraints

PRIMARY KEY (publication_id)

See also

- "sysarticle system table" on page 223

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(256)	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)

See also

- "syspublication system table" on page 223

sysuldata system table

The sysuldata system table stores schema information about the UltraLite database. In previous versions, this table stored database properties and options, however users should not use this table to query these values. Instead, use the **db_property()** function to get the value of specific properties or options: For name, the name or identifier of the property; for type, a value that indicates the type of property or data.

Column name	Column type	Description
long_setting	LONGBINARY	A BLOB for long values.
name	VARCHAR(128)	The name of the property.
setting	VARCHAR(128)	The value of the property.
type	VARCHAR(32)	One of either **sys** for internals, **opt** for options, or **prop** for properties

Constraints

PRIMARY KEY (name, type)

See also

- "UltraLite database properties" on page 158

UltraLite SQL reference

This section provides a reference for UltraLite SQL. UltraLite SQL is a unique subset of the SQL supported by SQL Anywhere databases.

UltraLite SQL elements

Keywords in UltraLite

Each SQL statement contains one or more keywords. SQL keywords are case insensitive, but throughout these manuals, keywords are indicated in uppercase. Some keywords cannot be used as identifiers without surrounding them in double quotes. These are called **reserved words**. See "Reserved words" [*SQL Anywhere Server - SQL Reference*].

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.

Identifiers in UltraLite

Identifiers are names of objects in the database, such as user IDs, tables, and columns. Identifiers have a maximum length of 128 bytes.

You must enclose identifiers in double quotes if any of the following conditions are true:

- The identifier contains spaces.
- The first character of the identifier is not an alphabetic character. The database collation sequence dictates which characters are considered alphabetic or digit characters.
- The identifier contains a reserved word. See "Reserved words" [*SQL Anywhere Server - SQL Reference*].
- The identifier contains characters other than alphabetic characters and digits.

You can only use a single backslash in an identifier if it is used as an escape character.

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.

See also

- "Strings" [*SQL Anywhere Server - SQL Reference*]
- "UltraLite character sets" on page 30

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 SQL/2003 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.

> **Note**
> The percent sign (%) is not supported in UltraLite.

Examples

- 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. */
```

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

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. See "NULL value" [*SQL Anywhere Server - SQL Reference*].

Special values in UltraLite

You can use special values in expressions, and as column defaults when you create tables.

CURRENT DATE special value

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

See also

- "Expressions in UltraLite" on page 246
- "GETDATE function [Date and time]" on page 304
- "NOW function [Date and time]" on page 329

CURRENT TIME special value

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

See also

- "Expressions in UltraLite" on page 246
- "GETDATE function [Date and time]" on page 304
- "NOW function [Date and time]" on page 329

CURRENT TIMESTAMP special value

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

See also

- “CURRENT TIME special value” on page 228
- “Expressions in UltraLite” on page 246
- “NOW function [Date and time]” on page 329
- “GETDATE function [Date and time]” on page 304
- “NOW function [Date and time]” on page 329

CURRENT UTC TIMESTAMP special value

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

See also

- "Expressions in UltraLite" on page 246
- "GETDATE function [Date and time]" on page 304
- "CURRENT TIMESTAMP special value" on page 228
- "NOW function [Date and time]" on page 329

SQLCODE special value

Current SQLCODE value at the time the special value was evaluated. UltraLite only (not supported for UltraLiteJ).

Data type

String

Remarks

The SQLCODE value is set after each statement. You can check the SQLCODE to determine if the statement succeeded.

See also

- "Expressions in UltraLite" on page 246
- "Error Messages"

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.

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. See "Specifying date parts" on page 267.

Data types in UltraLite

Available data types in UltraLite SQL include:

- Integer

- Decimal
- Floating-point
- Character
- Binary
- Date/time

Note
Domains (user-defined data types) are not supported in UltraLite SQL.

Note
You cannot concatenate LONGVARCHAR and LONGBINARY data types. See "String operators" on page 260.

You can create a host variable with any one of the supported types. UltraLite supports a subset of the data types available in SQL Anywhere. The following are the SQL data types supported in UltraLite databases.

Data type	Description
BIT	Boolean values (0 or 1). See "BIT data type" [*SQL Anywhere Server - SQL Reference*].
{ **CHAR** | **CHARACTER** } **(** *max-length* **)**	Character data of *max-length*, in the range of 1-32767 bytes. See "CHAR data type" [*SQL Anywhere Server - SQL Reference*]. When evaluating expressions, the maximum length for a temporary character value is 2048 bytes.
VARCHAR (*max-length* **)**	VARCHAR is used for variable-length character data of *max-length*. See "VARCHAR data type" [*SQL Anywhere Server - SQL Reference*].
LONG VARCHAR	Arbitrary length character data. Conditions in SQL statements (such as in the WHERE clause) cannot operate on LONG VARCHAR columns. The only operations allowed on LONG VARCHAR columns are to insert, update, or delete them, or to include them in the *select-list* of a query. See "LONG VARCHAR data type" [*SQL Anywhere Server - SQL Reference*]. You can cast strings to/from LONGVARCHAR data.
[**UNSIGNED**] **BIGINT**	An integer requiring 8 bytes of storage. See "BIGINT data type" [*SQL Anywhere Server - SQL Reference*].

Data type	Description
{ **DECIMAL** | **DEC** | **NUMERIC** } **(** *precision*, *scale*] **)**]	The representation of a decimal number using two parts: *precision* (total digits) and *scale* (digits that follow a decimal point). See "DECIMAL data type" [*SQL Anywhere Server - SQL Reference*], "NUMERIC data type" [*SQL Anywhere Server - SQL Reference*], "UltraLite precision creation parameter" on page 148, and "UltraLite scale creation parameter" on page 149.
DOUBLE [**PRECISION**]	A double-precision floating-point number. In this data type PRECISION is an optional part of the DOUBLE data type name. See "DOUBLE data type" [*SQL Anywhere Server - SQL Reference*].
FLOAT [**(** *precision* **)**]	A floating-point number, which may be single or double precision. See "FLOAT data type" [*SQL Anywhere Server - SQL Reference*].
[**UNSIGNED**] { **INT** | **INTEGER** }	An unsigned integer requiring 4 bytes of storage. See "INTEGER data type" [*SQL Anywhere Server - SQL Reference*].
REAL	A single-precision floating-point number stored in 4 bytes. See "REAL data type" [*SQL Anywhere Server - SQL Reference*].
[**UNSIGNED**] **SMALLINT**	An integer requiring 2 bytes of storage. See "SMALLINT data type" [*SQL Anywhere Server - SQL Reference*].
[**UNSIGNED**] **TINYINT**	An integer requiring 1 byte of storage. See "TINYINT data type" [*SQL Anywhere Server - SQL Reference*].
DATE	A calendar date, such as a year, month, and day. See "DATE data type" [*SQL Anywhere Server - SQL Reference*].
TIME	The time of day, containing hour, minute, second, and fraction of a second. See "TIME data type" [*SQL Anywhere Server - SQL Reference*].
DATETIME	Identical to TIMESTAMP. See "DATETIME data type" [*SQL Anywhere Server - SQL Reference*].
TIMESTAMP	A point in time, containing year, month, day, hour, minute, second, and fraction of a second. See "TIMESTAMP data type" [*SQL Anywhere Server - SQL Reference*].
VARBINARY (*max-length* **)**	Identical to BINARY. See "VARBINARY data type" [*SQL Anywhere Server - SQL Reference*].
BINARY (*max-length* **)**	Binary data of maximum length *max-length* bytes. The maximum length should not exceed 2048 bytes. See "BINARY data type" [*SQL Anywhere Server - SQL Reference*].

Data type	Description
LONG BINARY	Arbitrary length binary data. Conditions in SQL statements (such as in the WHERE clause) cannot operate on LONG BINARY columns. The only operations allowed on LONG BINARY columns are to insert, update, or delete them, or to include them in the *select-list* of a query. See "LONG BINARY data type" [*SQL Anywhere Server - SQL Reference*]. You can cast values to/from LONGBINARY data.
UNIQUEIDENTIFIER	Typically used for a primary key or other unique column to hold UUID (Universally Unique Identifier) values that uniquely identify rows. UltraLite provides functions that generate UUID values. These values are generated so that a value produced on one computer does not match a UUID produced on another computer. UNIQUEIDENTIFIER values generated in this way can therefore be used as keys in a synchronization environment. See "UNIQUEIDENTIFIER data type" [*SQL Anywhere Server - SQL Reference*].

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	NUMERIC(19,4)
SMALLMONEY	NUMERIC(10,4)
TEXT	LONG VARCHAR
XML	LONG VARCHAR

Converting data types explicitly

UltraLite allows you to request data type conversions explicitly, by using either the CAST or CONVERT function.

NOTE
Self-casting usually has no effect on operations. However, self-casts to CHAR/VARCHAR, BINARY/VARBINARY and NUMERIC are not no-op procedures.

You can CAST or CONVERT most combinations of data types, as illustrated by the table that follows.

The ability to convert or not is contingent upon the value used in the conversion. As the **Value-dependent** column shows, the value must be compatible with the new data type to avoid generating a specific type of conversion error. For example:

- If you cast **varchar "1234"** to **long**, this conversion is supported. However, if you cast **varchar "hello"** to **long**, then this conversion generates a **SQLE_CONVERSION_ERROR** error because **hello** is not a number.
- If you cast **long 1234** to **short**, this conversion is supported. However, if you cast **long 1000000** to **short**, then this conversion generates a **SQLE_OVERFLOW_ERROR** error, because **1000000** is beyond the range of numbers a short can hold.

From	Always	Never	Value-dependent
BINARY or VARBINARY	CHAR or VARCHAR	LONG VARCHAR	NUMERIC
	BINARY	REAL	UID[1]
	LONG BINARY	TIME	
	BIT	TIMESTAMP	
	TINYINT	DOUBLE	
	SHORT INT	DATE	
	SIGNED SHORT		
	BIGINT		
	SIGNED BIG		

From	Always	Never	Value-dependent
LONG BINARY	BINARY LONG BINARY	BIT CHAR or VARCHAR LONG VARCHAR TINYINT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIGINT SIGNED BIG REAL DOUBLE NUMERIC DATE TIME TIMESTAMP UID	N/A

From	Always	Never	Value-dependent
BIT	CHAR or VARCHAR BINARY BIT TINYINT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIGINT REAL SIGNED BIG DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	N/A

From	Always	Never	Value-dependent
CHAR or VAR-CHAR	BINARY or VARBINARY CHAR or VARCHAR LONG VARCHAR	LONG BINARY	BIT TINYINT SIGNED SHORT SHORT INT LONG INT SIGNED LONG BIGINT SIGNED BIG DOUBLE NUMERIC REAL DATE TIME TIMESTAMP UID

From	Always	Never	Value-dependent
LONG VAR-CHAR	CHAR or VARCHAR LONG VARCHAR	BINARY or VARBINARY LONG BINARY BIT TINYINT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIGINT SIGNED BIG REAL NUMERIC DATE TIME TIMESTAMP DOUBLE UID	

From	Always	Never	Value-dependent
TINYINT	BINARY or VARBINARY CHAR or VARCHAR TINYINT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIGINT SIGNED BIG REAL DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	
SHORT INT	BINARY or VARBINARY CHAR or VARCHAR SHORT INT LONG INT SIGNED LONG BIGINT SIGNED BIG REAL DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	BIT TINYINT SIGNED SHORT

From	Always	Never	Value-dependent
SIGNED SHORT	BINARY or VARBINARY CHAR or VARCHAR SIGNED SHORT SIGNED LONG SIGNED BIG REAL DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	SHORT INT LONG INT BIGINT BIT TINYINT
LONG INT	BINARY or VARBINARY CHAR or VARCHAR LONG INT BIGINT SIGNED BIG REAL DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	BIT TINYINT SHORT INT SIGNED SHORT SIGNED LONG
SIGNED LONG	BINARY or VARBINARY CHAR or VARCHAR SIGNED LONG SIGNED BIG REAL DOUBLE NUMERIC DATE TIMESTAMP	LONG VARCHAR LONG BINARY TIME UID	BIT TINYINT SHORT INT SIGNED SHORT LONG INT BIGINT

From	Always	Never	Value-dependent
BIGINT	BINARY or VARBINARY CHAR or VARCHAR BIGINT REAL DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	BIT TINYINT SHORT INT SIGNED SHORT LONG INT SIGNED LONG SIGNED BIG
SIGNED BIG	BINARY or VARBINARY CHAR or VARCHAR SIGNED BIG REAL DOUBLE NUMERIC DATE TIMESTAMP	LONG VARCHAR LONG BINARY TIME UID	BIT TINYINT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIGINT
REAL	CHAR or VARCHAR REAL DOUBLE NUMERIC	LONG VARCHAR BINARY or VARBINARY LONG BINARY DATE TIME TIMESTAMP UID	BIT TINYINT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIGINT SIGNED BIG

From	Always	Never	Value-dependent
DOUBLE	CHAR or VARCHAR DOUBLE NUMERIC	LONG VARCHAR BINARY or VARBINARY LONG BINARY DATE TIME TIMESTAMP UID	BIT TINYINT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIGINT SIGNED BIG REAL
NUMERIC	CHAR or VARCHAR REAL NUMERIC DOUBLE	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	BINARY or VARBINARY[2] BIT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIGINT SIGNED BIG TINYINT

From	Always	Never	Value-dependent
DATE	CHAR or VARCHAR	LONG VARCHAR	
	SIGNED LONG	LONG BINARY	
	SIGNED BIG	BIT	
	DATE	TINYINT	
	TIMESTAMP	SHORT INT	
		SIGNED SHORT	
		LONG INT	
		BIGINT	
		REAL	
		DOUBLE	
		NUMERIC	
		TIME	
		BINARY or VARBINARY	
		UID	

From	Always	Never	Value-dependent
TIME	CHAR or VARCHAR TIME TIMESTAMP	LONG VARCHAR LONG BINARY BIT TINYINT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIGINT SIGNED BIG REAL DOUBLE NUMERIC DATE BINARY or VARBINARY UID	

From	Always	Never	Value-dependent
TIME-STAMP	CHAR or VARCHAR SIGNED LONG SIGNED BIG DATE TIME TIMESTAMP	LONG VARCHAR LONG BINARY BIT TINYINT SHORT INT SIGNED SHORT LONG INT BIGINT REAL DOUBLE NUMERIC BINARY or VARBINARY UID	

From	Always	Never	Value-dependent
UID	CHAR or VARCHAR UID	LONG VARCHAR LONG BINARY BIT TINYINT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIGINT SIGNED BIG REAL DOUBLE NUMERIC DATE TIME TIMESTAMP	BINARY or VARBINARY[1]

[1] The BINARY value must be a 16 byte length to be compatible with a UUID.

[2] Only works if the source NUMERIC value is able to cast as a BIGINT.

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:
{ **+** | **-** | ***** | **/** | **||** | **%** }

See also

- “Constants in expressions” on page 248
- “Special values in UltraLite” on page 227
- “Column names in expressions” on page 248
- “UltraLite SQL functions” on page 266
- “Subqueries in expressions” on page 251
- “Search conditions in UltraLite” on page 253
- “Data types in UltraLite” on page 230
- “CASE expressions” on page 249
- “Input parameters” on page 251

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 ('').

See also

- "Escape sequences" [*SQL Anywhere Server - SQL Reference*]

Example

To use a possessive phrase, type the string literal as follows:

'John''s database'

Column names in expressions

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"
```

See also

- "FROM clause [UltraLite]" on page 395

IF expressions

Sets a search condition to return a specific subset of data.

Syntax 1

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.

See also

- "NULL value" [*SQL Anywhere Server - SQL Reference*]
- "Search conditions" [*SQL Anywhere Server - SQL Reference*]

CASE expressions

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;
```

Aggregate expressions

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;
```

Subqueries in expressions

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).

See also

- "SELECT statement [UltraLite] [UltraLiteJ]" on page 402
- "Using subqueries" [*SQL Anywhere Server - SQL Usage*]
- "Search conditions in UltraLite" on page 253

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 );
```

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.

See also

- "Using host variables" [*UltraLite - C and C++ Programming*]
- "Preparing statements" [*SQL Anywhere Server - Programming*]
- UltraLite C/C++: "Data manipulation: Insert, Delete, and Update" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "Data manipulation: INSERT, UPDATE, and DELETE" [*UltraLite - .NET Programming*]
- UltraLite for M-Business: "Data manipulation: INSERT, UPDATE, and DELETE" [*UltraLite - M-Business Anywhere Programming*]

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.

Search conditions in UltraLite

Specifies a search condition 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 compare expression
| *expression* **IS** [**NOT**] { **NULL** | **TRUE** | **FALSE** | **UNKNOWN** }
| *expression* [**NOT**] **BETWEEN** *expression* **AND** *expression*
| *expression* [**NOT**] **IN (** *expression*, … **)**
| *expression* [**NOT**] **IN (** *subquery* **)**
| *expression* [**NOT**] { **ANY** | **ALL** } **(** *subquery* **)**
| *expression* [**NOT**] **EXISTS (** *subquery* **)**
| **NOT** *search-condition*
| *search-condition* **AND** *search-condition*
| *search-condition* **OR** *search-condition*
| (*search-condition*)

Parameters

compare:
= | > | < | >= | <= | <> | != | !< | !>

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.

The different types of search conditions supported by UltraLite include:

- ALL conditions
- ANY conditions
- BETWEEN conditions
- EXISTS conditions
- IN conditions

These conditions are discussed in separate sections that follow.

Note
Subqueries form an important class of expression that is used in many search conditions.

See also

- "Comparison operators" on page 254
- "Three-valued logic" [*SQL Anywhere Server - SQL Reference*]
- "Subqueries in expressions" on page 251

Comparison operators

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
[**NOT**] **LIKE**	a text comparison, possibly using regular expressions
>	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.

See also

- "Logical operators" on page 255
- "Search conditions in UltraLite" on page 253

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;
```

Logical operators

Does any of the following:

- Compare conditions (AND, OR, and NOT).
- Test the truth or NULL value nature of the expressions (IS).

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 **OR** *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.

See also

- "Three-valued logic" [*SQL Anywhere Server - SQL Reference*]
- "Comparison operators" on page 254
- "Search conditions in UltraLite" on page 253

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.

ALL conditions

Use the ALL condition in conjunction with a 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.

See also

- "Subqueries and the ALL test" [*SQL Anywhere Server - SQL Usage*]
- "Comparison operators" on page 254

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);
```

ANY conditions

Use the ANY condition in conjunction with a 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.

See also

- "Subqueries and the ANY test" [*SQL Anywhere Server - SQL Usage*]
- "Comparison operators" on page 254

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);
```

BETWEEN conditions

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 cheaper than $10 or more expensive than $15.

```
SELECT Name, UnitPrice
FROM Products
WHERE UnitPrice NOT BETWEEN 10 AND 15;
```

EXISTS conditions

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));
```


IN conditions

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');
```

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.

See also

- “Operator precedence” on page 261
- “Comparison operators” on page 254
- “Arithmetic operators” on page 260
- “String operators” on page 260
- “Bitwise operators” on page 261
- “Logical operators” on page 255

Arithmetic operators

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.

See also

- “Arithmetic operations” [*SQL Anywhere Server - SQL Usage*]

String operators

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.

Bitwise operators

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;
```

Operator precedence

The precedence of operators in expressions is as follows. 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.

Tip
Make the order of operation explicit in UltraLite, rather than relying on an operator precedence. That means, when you use more than one operator in an expression you should order operations clearly with parentheses.

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

Variables in UltraLite

You cannot use SQL variables (including global variables) in UltraLite applications.

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.

Performance tip
If you are not going to update data with the query, you should specify the FOR READ ONLY clause in your query. This clause may yield better performance. See "SELECT statement [UltraLite] [UltraLiteJ]" on page 402.

When to view an execution plan

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. See "UltraLite temporary files" on page 10.
- 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.

View an UltraLite execution plan

As a development aid, you can 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.

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.

To view an execution plan in the Plan Viewer

1. Choose **Tools** » **Plan Viewer**.
2. In the **SQL** pane, type a query.
3. Click **Get Plan** to generate a plan for the specified SQL statements.

Example

The text plan appears in the lower pane of the Plan Viewer.

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 index named primary). It then uses the index named secondary from the Transactions table to read only the row whose inv_no column matches.

See also

- "Interactive SQL utility (dbisql)" [*SQL Anywhere Server - Database Administration*]
- "Reading UltraLite execution plans" on page 264

Reading UltraLite execution plans

Because UltraLite short plans are textual summaries of how a query is accessed, you need to understand how the operations of either a join or a scan of a table are implemented.

- **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
count(*)	Counts the number of rows in a table.
distinct[*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.
dummy	No operation performed. It only occurs in two cases: • When you specify DUMMY in a FROM clause. • When the FROM clause is missing from the query.

Operation	Description
filter[*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.
group-by[*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.
group-single[*plan*]	Creates an aggregate of GROUP BY results, but only when it is known that a single row will be returned.
keyset[*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.
index-scan(*table-name*, *index-name*)	Reads only part of the table; the index is used to find the starting row.
join[*plan*, *plan*]	Performs an inner join between two plans.
lojoin[*plan*, *plan*]	Performs a left outer join between two plans.
like-scan(*table-name*, *index-name*)	Reads only part of a table; the index is used to find the starting row by pattern matching.
rowlimit[*plan*]	Performs the row limiting operation on propagated rows. Row limits are set by the TOP n or FIRST clause of the SELECT statement.
scan(*table-name*, *index-name*)	Reads an entire table following the order indicated by the index.
sub-query[*plan*]	Marks the start of a subquery.
temp[*plan*]	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.

Operation	Description
union-all[*plan*, ..., *plan*]	Performs a UNION ALL operation on the rows generated in the underlying plan.

UltraLite SQL functions

Functions are used to return information from the database. They are allowed 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. For a complete list of syntax conventions, see "Syntax conventions" [*SQL Anywhere Server - SQL Reference*].

Function types

This section groups the available function by type.

UltraLite supports a subset of the same functions documented for SQL Anywhere, and sometimes with a few differences.

See "Functions for spatial data" on page 415 for information on the new UltraLite spatial functions.

Note
Unless otherwise stated, any function that receives NULL as a parameter returns NULL.

UltraLite 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 functions

The following aggregate functions are available:

- “AVG function [Aggregate]” on page 277
- “COUNT function [Aggregate]” on page 288
- “COUNT_UPLOAD_ROWS function [Aggregate]” on page 289
- “LIST function [Aggregate]” on page 315
- “MAX function [Aggregate]” on page 320
- “MIN function [Aggregate]” on page 321
- “SUM function [Aggregate]” on page 353

UltraLite 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 functions

The following data type conversion functions are available:

- “CAST function [Data type conversion]” on page 279
- “CONVERT function [Data type conversion]” on page 285
- “HEXTOINT function [Data type conversion]” on page 306
- “INTTOHEX function [Data type conversion]” on page 310
- “ISDATE function [Data type conversion]” on page 311

UltraLite 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 SQL Anywhere TIMESTAMP data type.

For more information about datetime data types, see “Data types in UltraLite” on page 230.

Specifying date parts

Many of the date functions use dates built from **date parts**. The following table displays allowed values of date parts.

When using date and time functions, 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());
```

Date part	Abbreviation	Values
Year	yy	1-9999
Quarter	qq	1-4
Month	mm	1-12
Week	wk	1-54. Weeks begin on Sunday.
Day	dd	1-31
Dayofyear	dy	1-366
Weekday	dw	1-7 (Sunday = 1, ..., Saturday = 7)
Hour	hh	0-23
Minute	mi	0-59
Second	ss	0-59
Millisecond	ms	0-999
Microsecond	mcs or us	0-999999
Calyearofweek	cyr	Integer. The year in which the week begins. The week containing the first few days of the year may have started in the previous year, depending on the weekday on which the year started. Years starting on Monday through Thursday have no days that are part of the previous year, but years starting on Friday through Sunday start their first week on the first Monday of the 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 http://en.wikipedia.org/wiki/ISO_week_date.
Caldayofweek	cdw	1-7. (Monday = 1, ..., Sunday = 7)
TZ Offset	tz	-840 to 840

List of date and time functions

The following date and time functions are available:

UltraLite miscellaneous functions

Miscellaneous functions perform operations on arithmetic, string, or date/time expressions, including the return values of other functions.

List of functions

The following miscellaneous functions are available:

- "ARGN function [Miscellaneous]" on page 273
- "COALESCE function [Miscellaneous]" on page 284
- "EXPLANATION function [Miscellaneous]" on page 303
- "GREATER function [Miscellaneous]" on page 305
- "IFNULL function [Miscellaneous]" on page 309
- "ISNULL function [Miscellaneous]" on page 311
- "LESSER function [Miscellaneous]" on page 314
- "NEWID function [Miscellaneous]" on page 328
- "NULLIF function [Miscellaneous]" on page 330

UltraLite numeric functions

Numeric functions perform mathematical operations on numerical data types or return numeric information.

List of functions

The following numeric functions are available:

- "ABS function [Numeric]" on page 272
- "ACOS function [Numeric]" on page 273
- "ASIN function [Numeric]" on page 275
- "ATAN function [Numeric]" on page 275
- "ATAN2 function [Numeric]" on page 276
- "CEILING function [Numeric]" on page 280
- "COS function [Numeric]" on page 287
- "COT function [Numeric]" on page 288
- "DEGREES function [Numeric]" on page 300
- "EXP function [Numeric]" on page 302
- "FLOOR function [Numeric]" on page 304
- "LOG function [Numeric]" on page 317
- "LOG10 function [Numeric]" on page 318
- "MOD function [Numeric]" on page 325
- "PI function [Numeric]" [*SQL Anywhere Server - SQL Reference*]
- "POWER function [Numeric]" on page 332
- "RADIANS function [Numeric]" on page 334
- "REMAINDER function [Numeric]" on page 336
- "ROUND function [Numeric]" on page 340
- "SIGN function [Numeric]" on page 344
- "SIN function [Numeric]" on page 346
- "SQRT function [Numeric]" [*SQL Anywhere Server - SQL Reference*]
- "TAN function [Numeric]" on page 356
- "TRUNCNUM function [Numeric]" on page 359

UltraLite 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 functions

The following string functions are available:

UltraLite system functions

System functions return system information.

List of functions

The following system functions are available in UltraLite:

- "DB_PROPERTY function [System]" on page 299
- "ML_GET_SERVER_NOTIFICATION [System]" on page 324
- "SYNC_PROFILE_OPTION_VALUE function [System]" on page 355

Functions

Each function is listed, and the function type (numeric, character, and so on) is indicated next to it.

For links to all functions of a given type, see "Function types" on page 266.

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.

Numeric-expression data type	Returns
INT	INT
FLOAT	FLOAT
DOUBLE	DOUBLE
NUMERIC	NUMERIC

Standards and compatibility

- **SQL/2008** The ABS function is part of optional SQL/2008 language feature T441.

Example

The following statement returns the value 66.

```
SELECT ABS( -66 );
```


ACOS function [Numeric]

Returns the arc-cosine, in radians, of a numeric expression. Supported in UltraLite but not UltraLiteJ.

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.

See also

- "ASIN function [Numeric]" on page 275
- "ATAN function [Numeric]" on page 275
- "ATAN2 function [Numeric]" on page 276
- "COS function [Numeric]" on page 287

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the arc-cosine value for 0.52.

```
SELECT ACOS( 0.52 );
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 6.

```
SELECT ARGN( 6, 1,2,3,4,5,6 );
```

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.

See also

- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 90.

```
SELECT ASCII( 'Z' );
```


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.

See also

- "ACOS function [Numeric]" on page 273
- "ATAN function [Numeric]" on page 275
- "ATAN2 function [Numeric]" on page 276
- "SIN function [Numeric]" on page 346

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the arc-sine value for 0.52.

```
SELECT ASIN( 0.52 );
```

ATAN function [Numeric]

Returns the arc-tangent, in radians, of a number. Supported in UltraLite but not UltraLiteJ.

Syntax

ATAN(*numeric-expression* **)**

Remarks

The ATAN and TAN functions are inverse operations.

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.

See also

- "ACOS function [Numeric]" on page 273
- "ASIN function [Numeric]" on page 275
- "ATAN2 function [Numeric]" on page 276
- "TAN function [Numeric]" on page 356

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the arc-tangent value for 0.52.

```
SELECT ATAN( 0.52 );
```

ATAN2 function [Numeric]

Returns the arc-tangent, in radians, of the ratio of two numbers. Supported in UltraLite but not UltraLiteJ.

Syntax

ATAN2 (*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.

See also

- "ACOS function [Numeric]" on page 273
- "ASIN function [Numeric]" on page 275
- "ATAN function [Numeric]" on page 275
- "TAN function [Numeric]" on page 356

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the arc-tangent value for the ratio 0.52 to 0.60.

```
SELECT ATAN2( 0.52, 0.60 );
```

AVG function [Aggregate]

Computes the average, for a set of rows, of a numeric expression or of a set of unique values.

Syntax 1

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. See "CAST function [Data type conversion]" [*SQL Anywhere Server - SQL Reference*].

See also

- "SUM function [Aggregate]" on page 353
- "COUNT function [Aggregate]" on page 288

Standards and compatibility

- **SQL/2008** AVG is a core feature of the SQL/2008 standard.

Example

The following statement returns the value 49988.623200.

```
SELECT AVG( Salary ) FROM Employees;
```

The following statement returns the average product price from the Products table:

```
SELECT AVG( DISTINCT UnitPrice ) FROM Products;
```

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.

See also

- "CHAR_LENGTH function [String]" on page 282
- "DATALENGTH function [System]" on page 290
- "LENGTH function [String]" on page 314
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension. The equivalent function in the SQL/2008 standard is the OCTET_LENGTH function.

Example

The following statement returns the value 12.

```
SELECT BYTE_LENGTH( 'Test Message' );
```


BYTE_SUBSTR function [String]

Returns a substring of a string. The substring is calculated using bytes, not characters.

Syntax

BYTE_SUBSTR(*string-expression*, *start* [, *length*] **)**

Parameters

- **string-expression** The string from which the substring is taken.
- **start** An integer expression indicating the start of the substring. A positive integer starts from the beginning of the string, with the first character being position 1. A negative integer specifies a substring starting from the end of the string, the final character 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, VARCHAR, or NVARCHAR. The value returned depends on the type of *string-expression*. Also, the arguments you specify determine if the returned value is LONG. For example, LONG is not returned when you specify a constant < 32K for length.

Remarks

If *length* is specified, the substring is restricted to that number of bytes. Both *start* and *length* can be either positive or negative. Using appropriate combinations of negative and positive numbers, you can get a substring from either the beginning or end of the string.

If *start* is zero and length is non-negative, a *start* value of 1 is used. If *start* is zero and *length* is negative, a start value of -1 is used.

See also

- "SUBSTRING function [String]" on page 352
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value Test.

```
SELECT BYTE_SUBSTR( 'Test Message', 1, 4 );
```

CAST function [Data type conversion]

Returns the value of an expression converted to a supplied data type.

The CAST, CONVERT, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values. For more information on using these functions, see "Converting to and from hexadecimal values" [*SQL Anywhere Server - SQL Reference*].

Syntax

CAST(*expression* **AS** *datatype* **)**

Parameters

- **expression** The expression to be converted.
- **data type** The target data type.

Returns

Depends on the data type requested.

Remarks

If you do not indicate a length for character string types, the database server chooses an appropriate length. If neither precision nor scale is specified for a DECIMAL conversion, the database server selects appropriate values.

See also

- "Data type conversions" [*SQL Anywhere Server - SQL Reference*]
- "CONVERT function [Data type conversion]" on page 285
- "LEFT function [String]" on page 313

Standards and compatibility

- **SQL/2008** The CAST function is a core feature of the SQL/2008 standard. However, in SQL Anywhere CAST supports a number of data type conversions that are not permitted by the SQL standard. For example, in SQL Anywhere you can CAST an integer value to a DATE type, whereas in the SQL standard this type conversion is not permitted. For more information, see "Data type conversions" [*SQL Anywhere Server - SQL Reference*].

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 );
```

You can use the CAST function to shorten strings

```
SELECT CAST ( 'Surname' AS CHAR(5) );
```

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.

See also

- "FLOOR function [Numeric]" on page 304

Standards and compatibility

- **SQL/2008** The CEILING function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns the value 60.

```
SELECT CEILING( 59.84567 );
```

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.

See also

- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value Y.

```
SELECT CHAR( 89 );
```

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.

> **Note**
> 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.

See also

- "BYTE_LENGTH function [String]" on page 278
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** CHAR_LENGTH is a core feature of the SQL/2008 standard. Using CHAR_LENGTH over an expression of type NCHAR comprises part of optional SQL/2008 language feature F421.

Example

The following statement returns the value 8.

```
SELECT CHAR_LENGTH( 'Chemical' );
```

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.
- **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.

See also

- "SUBSTRING function [String]" on page 352
- "REPLACE function [String]" on page 337
- "LOCATE function [String]" on page 316
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

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 includes the letter K:

```
SELECT Surname, GivenName
FROM Employees
WHERE CHARINDEX( 'K', Surname ) = 1;
```

Results returned:

Surname	GivenName
Klobucher	James
Kuo	Felicia
Kelly	Moira

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

ANY

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.

For a more detailed description of how the database server processes this function, see "ISNULL function [Miscellaneous]" on page 311.

See also

- "ISNULL function [Miscellaneous]" on page 311

Standards and compatibility

- **SQL/2008** Core feature.

Example

The following statement returns the value 34.

```
SELECT COALESCE( NULL, 34, 13, 0 );
```

CONVERT function [Data type conversion]

Returns an expression converted to a supplied data type.

The CAST, CONVERT, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values. For more information on using these functions, see "Converting to and from hexadecimal values" [*SQL Anywhere Server - SQL Reference*].

Syntax

CONVERT(*datatype*, *expression* [, *format-style*] **)**

Parameters

- **datatype** The data type to which the expression is converted.
- **expression** The expression to be converted.
- **format-style** The style code to apply to the outputted 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).

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

Without century (yy) style codes	With century (yyyy) style codes	Output format
-	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

If no *format-style* argument is provided, style code 0 is used.

For a description of the styles produced by each output symbol (such as Mmm), see "UltraLite date_format creation parameter" on page 138.

See also

- "CAST function [Data type conversion]" on page 279
- "CSCONVERT function [String]" [*SQL Anywhere Server - SQL Reference*]

Standards and compatibility

- **SQL/2008** Vendor extension. The CONVERT function is defined in the SQL/2008 standard. However, in the SQL standard the purpose of CONVERT is to perform a transcoding of the input string expression to a different character set, which is implemented in SQL Anywhere as the CSCONVERT function.

Example

The following statements illustrate the use of format style.

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 104 ) FROM SalesOrders;
```


OrderDate
16.03.2000
20.03.2000
23.03.2000
25.03.2000
...

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 7 ) FROM 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 );
```

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.

See also

- "ACOS function [Numeric]" on page 273
- "COT function [Numeric]" on page 288
- "SIN function [Numeric]" on page 346
- "TAN function [Numeric]" on page 356

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value of the cosine of an angle 0.52 radians.

```
SELECT COS( 0.52 );
```

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.

See also

- "COS function [Numeric]" on page 287
- "SIN function [Numeric]" on page 346
- "TAN function [Numeric]" on page 356

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the cotangent value of 0.52.

```
SELECT COT( 0.52 );
```

COUNT function [Aggregate]

Counts the number of rows in a group depending on the specified parameters.

Syntax 1

COUNT([* | [**DISTINCT**] *expression*] **)**

Parameters

- ***** Return the number of rows in each group. COUNT(*) and COUNT() are semantically equivalent.
- **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.

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

The COUNT function returns a maximum value of 2147483647.

See also

- "AVG function [Aggregate]" on page 277
- "SUM function [Aggregate]" on page 353

Standards and compatibility

- **SQL/2008** Core feature.

Example

The following statement returns each unique city, and the number of employees working in that city.

```
SELECT City, COUNT( * ) FROM Employees GROUP BY City;
```

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

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following returns the total number of rows to upload in *mypub1* and *mypub2*:

```
SELECT COUNT_UPLOAD_ROWS('mypub1,mypub2', 0);
```

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
SMALLINT	2
INTEGER	4
DOUBLE	8
CHAR	Length of the data
BINARY	Length of the data

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the length of the longest string in the CompanyName column.

```
SELECT MAX( DATALENGTH( CompanyName ) )
FROM Customers;
```

The following statement returns the length of the string '8sdofinsv8s7a7s7gehe4h':


```
SELECT DATALENGTH( '8sdofinsv8s7a7s7gehe4h' );
```

DATE function [Date and time]

Converts the expression into a date, and removes any hours, minutes, or seconds.

For information about controlling the interpretation of date formats, see "UltraLite date_order creation parameter" on page 141.

Syntax

DATE(*expression* **)**

Returns

DATE

Parameters

- **expression** The value to be converted to date format, typically a string.

Standards and compatibility

- **SQL/2008** Vendor extension.

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 SYSOBJECT;
```

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.

  For a complete listing of allowed date parts, see "Specifying date parts" on page 267.

- **integer-expression** The number of *date-part* values to be added to *timestamp-expression*. Note that *integer-expression* can be any numeric type, but its value is truncated to an INTEGER.

- **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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the TIMESTAMP value 1995-11-02 00:00:00.000.

```
SELECT DATEADD( month, 102, '1987/05/02' );
```

The following statement returns the TIMESTAMP value 1987-05-02 04:00:00.000.

```
SELECT DATEADD( hour, 4, '1987/05/02' );
```

The following statement returns the TIMESTAMP WITH TIME ZONE value 1987-05-06 11:33:00.000+04:00

```
SELECT DATEADD( day, 4, CAST( '1987/05/02 11:33:00.000000+04:00' as TIMESTAMP
WITH TIME ZONE ));
```

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.

  Choose one of the date objects listed above. For a complete list of date parts, see "Specifying date parts" on page 267.

- **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

Remarks

This function calculates the number of date parts between two specified dates. The result is a signed integer value equal to (date2 - date1), 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 and compatibility

- **SQL/2008** Vendor extension.

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' );
```

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.

  For information about date format descriptions, see "UltraLite date_format creation parameter" on page 138.

Returns

VARCHAR

Remarks

Any allowable date format can be used for the string-expression.

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value Jan 01, 1989.

```
SELECT DATEFORMAT( '1989-01-01', 'Mmm dd, yyyy' );
```

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.

  For a complete listing of allowed date parts, see "Specifying date parts" on page 267.

- **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.

See also

- "DATEPART function [Date and time]" on page 295

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value May.

```
SELECT DATENAME( month, '1987/05/02' );
```

DATEPART function [Date and time]

Syntax

DATEPART(*date-part*, *timestamp-expression* **)**

Parameters

- **date-part** The date part to be returned.

  For a complete listing of allowed date parts, see "Specifying date parts" on page 267.

- **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.

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 5.

```
SELECT DATEPART( month , '1987/05/02' );
```

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 SalesOrders
   GROUP BY DATEPART( Year, OrderDate );
```

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.

See also

- "CAST function [Data type conversion]" on page 279

Standards and compatibility

- **SQL/2008** Vendor extension.

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' );
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 12.

```
SELECT DAY( '2001-09-12' );
```

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 English names are returned as: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday.

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value Saturday.

```
SELECT DAYNAME ( '1987/05/02' );
```

DAYS function [Date and time]

The DAYS function manipulates a TIMESTAMP, or returns the number of days between two TIMESTAMP values. For specific details, see the Remarks section below.

Syntax 1

DAYS(*timestamp-expression* **)**

Syntax 2

DAYS(*timestamp-expression*, *timestamp-expression* **)**

Syntax 3

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.

  For information about casting data types, see "CAST function [Data type conversion]" on page 279.

Returns

INTEGER with Syntax 1 or Syntax 2.

TIMESTAMP with Syntax 3.

Remarks

The result of the DAYS function depends on its arguments. The DAYS function ignores hours, minutes, and seconds in its arguments.

- **Syntax 1** 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.

Note
0000-02-29 is not meant to imply an actual date; it is the default date used by the DAYS function.

- **Syntax 2** If you pass two TIMESTAMP values to the DAYS function, the function returns the integer number of days between them.

- **Syntax 3** 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.

Instead of Syntax 2, use the DATEDIFF function. Instead of Syntax 3, use the DATEADD function.

See also

- "DATEDIFF function [Date and time]" on page 292
- "DATEADD function [Date and time]" on page 291

Standards and compatibility

- **SQL/2008** Vendor extension.

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' );
```

DB_PROPERTY function [System]

Returns the value of the given property. Supported in UltraLite but not UltraLiteJ.

Syntax

DB_PROPERTY(*property-name* **)**

Parameters

- **property-name** The database property name.

Returns

VARCHAR, LONG VARCHAR

Remarks

Returns a string.

To set an option in UltraLite, use the SET OPTION statement or your component's API-specific Set Database Option method.

See also

- "SET OPTION statement [UltraLite] [UltraLiteJ]" on page 404
- UltraLite C++: "SetDatabaseOption method" [*UltraLite - C and C++ Programming*]
- UltraLite.NET: "SetDatabaseOption method" [*UltraLite - .NET Programming*]

Example

The following statement returns the page size of the current database, in bytes.

```
SELECT DB_PROPERTY( 'page_size');
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 29.79380534680281.


```
SELECT DEGREES( 0.52 );
```

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.

See also

- "SOUNDEX function [String]" on page 347
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns similarity between the words test and chest:

```
SELECT DIFFERENCE( 'test', 'chest' );
```

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 and compatibility

- **SQL/2008** Vendor extension.

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 employees StartDate, expressed as the number of the day of the week:

```
SELECT DOW( StartDate ) FROM Employees;
```

EXP function [Numeric]

Returns the result of the base of natural logarithms e raised to the power of the given argument (not supported in UltraLiteJ).

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 and compatibility

- **SQL/2008** The EXP function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The statement returns the value 3269017.3724721107.

```
SELECT EXP( 15 );
```

EXPLANATION function [Miscellaneous]

Returns the optimization strategy of an SQL statement as a plain text string.

Syntax

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.

Returns

LONG VARCHAR

Remarks

The statement's access plan is returned as a string. To interpret the result, see "Reading execution plans" [*SQL Anywhere Server - SQL Usage*]. 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.

See also

- "Execution plans in UltraLite" on page 262

Standards and compatibility

- **SQL/2008** Vendor extension.

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' );
```

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.

See also

- "CEILING function [Numeric]" on page 280

Standards and compatibility

- **SQL/2008** The FLOOR function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns a Floor value of 123:

```
SELECT FLOOR (123);
```

The following statement returns a value of 123:

```
SELECT FLOOR (123.45);
```

The following statement returns a value of -124:

```
SELECT FLOOR (-123.45);
```

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.

See also

- "NOW function [Date and time]" on page 329

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the system date and time.

```
SELECT GETDATE( );
```

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

Depends on the parameters that are compared.

Remarks

If the parameters are equal, the first is returned.

See also

- "LESSER function [Miscellaneous]" on page 314

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 10.

```
SELECT GREATER( 10, 5 ) FROM dummy;
```

HEXTOINT function [Data type conversion]

Returns the decimal integer equivalent of a hexadecimal string.

The CAST, CONVERT, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values. For more information on using these functions, see "Converting to and from hexadecimal values" [*SQL Anywhere Server - SQL Reference*].

Syntax

HEXTOINT(*hexadecimal-string* **)**

Parameters

- **hexadecimal-string** The string to be converted to an integer.

Returns

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 literals 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.

See also

- "INTTOHEX function [Data type conversion]" on page 310

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 420.


```
SELECT HEXTOINT( '1A4' );
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 21:

```
SELECT HOUR( '1998-07-09 21:12:13' );
```

HOURS function [Date and time]

The HOURS function manipulates a TIMESTAMP, or returns the number of hours between two TIMESTAMP values. For specific details, see this function's usage.

Syntax 1

HOURS (*timestamp-expression* **)**

Syntax 2

HOURS (*timestamp-expression*, *timestamp-expression* **)**

Syntax 3

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*..

  For information about casting data types, see "CAST function [Data type conversion]" on page 279.

Returns

INTEGER with Syntax 1 or Syntax 2.

TIME or TIMESTAMP with Syntax 3.

Remarks

The result of the HOURS function depends on its arguments.

- **Syntax 1** 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.

  Note
  0000-02-29 is not meant to imply an actual date; it is the default TIMESTAMP value used by the HOURS function.

- **Syntax 2** If you pass two TIMESTAMP values to the HOURS function, the function returns the integer number of hours between them.
- **Syntax 3** 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. Syntax 3 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.

Instead of Syntax 2, use the DATEDIFF function. Instead of Syntax 3, use the DATEADD function.

See also

- "DATEDIFF function [Date and time]" on page 292
- "DATEADD function [Date and time]" on page 291

Standards and compatibility

- **SQL/2008** Vendor extension.

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' );
```

IFNULL function [Miscellaneous]

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.

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*.

Standards and compatibility

- **SQL/2008** Vendor extension.

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 );
```

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 VARCHAR

See also

- "STUFF function [String]" on page 351
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value backoffice.

```
SELECT INSERTSTR( 0, 'office ', 'back' );
```

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. For more information, see "Converting to and from hexadecimal values" [*SQL Anywhere Server - SQL Reference*].

See also

- "HEXTOINT function [Data type conversion]" on page 306

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 0000009c.

```
SELECT INTTOHEX( 156 );
```

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.

Standards and compatibility

- **SQL/2008** Vendor extension.

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 result in the type used for the comparison. If the database server cannot find a common comparison type, an error is returned.

See also

- "COALESCE function [Miscellaneous]" on page 284

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value -66.

```
SELECT ISNULL( NULL ,-66, 55, 45, NULL, 16 );
```

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

- CHAR
- NCHAR
- LONG VARCHAR
- VARCHAR
- NVARCHAR

Remarks

The LCASE function is identical to the LOWER function.

See also

- "LOWER function [String]" on page 318
- "UCASE function [String]" on page 360
- "UPPER function [String]" on page 360
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension. The equivalent function LOWER is a core feature of the SQL/2008 standard.

Example

The following statement returns the value chocolate.

```
SELECT LCASE( 'ChoCOlatE' );
```

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
- LONG NVARCHAR

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.

See also

- “RIGHT function [String]” on page 339
- “UltraLite string functions” on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the first 5 characters of each Surname value in the Customers table.

```
SELECT LEFT( Surname, 5) FROM Customers;
```

LENGTH function [String]

Returns the number of characters in the specified string.

Syntax

LENGTH(*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.

> **Note**
> You can use the LENGTH function and the CHAR_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.

See also

- "BYTE_LENGTH function [String]" on page 278
- "International languages and character sets" [*SQL Anywhere Server - Database Administration*]
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** The LENGTH function is a vendor extension; however, its semantics are identical to that of the CHAR_LENGTH function in the SQL/2008 standard. Using LENGTH over a string expression of type NCHAR comprises part of optional SQL/2008 language feature F421.

Example

The following statement returns the value 9.

```
SELECT LENGTH( 'chocolate' );
```

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.

See also

- "GREATER function [Miscellaneous]" on page 305

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 5.

```
SELECT LESSER( 10, 5 ) FROM dummy;
```

LIST function [Aggregate]

Returns a delimited list of values for every row in a group.

Syntax

LIST(
[**DISTINCT**] *string-expression*
[, *delimiter-string*] **)**

Parameters

- **string-expression** A string expression, usually a column name. 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.
- **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.

Returns

- LONG VARCHAR
- 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.

Standards and compatibility

- **SQL/2008** Vendor extension.

Examples

The following statement returns all street addresses from the Employees table.

```
SELECT LIST( Street ) FROM Employees;
```

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.
- **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 * 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 255 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.

See also

- "UltraLite string functions" on page 271
- "CHARINDEX function [String]" on page 283

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 8.

```
SELECT LOCATE(
   'office party this week - rsvp as soon as possible',
   'party',
   2 );
```

LOG function [Numeric]

Returns the natural logarithm of a number (not supported in UltraLiteJ).

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.

See also

- "LOG10 function [Numeric]" on page 318

Standards and compatibility

- **SQL/2008** The SQL/2008 standard defines the natural logarithm function using the keyword LN. The natural logarithm function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns the natural logarithm of 50.

```
SELECT LOG( 50 );
```

LOG10 function [Numeric]

Returns the base 10 logarithm of a number (not supported in UltraLiteJ).

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.

See also

- "LOG function [Numeric]" on page 317

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the base 10 logarithm for 50.

```
SELECT LOG10( 50 );
```

LOWER function [String]

Converts all characters in a string to lowercase. This function is identical to the LCASE function.

Syntax

LOWER(*string-expression* **)**

Parameters

- **string-expression** The string to be converted to lowercase.

Returns

CHAR, VARCHAR, LONG VARCHAR, NCHAR, NVARCHAR, or LONG NVARCHAR corresponding to the data type of the argument.

Remarks

The LCASE function is identical to the LOWER function.

See also

- "LCASE function [String]" on page 312
- "UCASE function [String]" on page 360
- "UPPER function [String]" on page 360
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** The LOWER function is a core feature of the SQL/2008 standard. Using LOWER over an expression of type NCHAR comprises part of optional SQL/2008 language feature F421.

Example

The following statement returns the value chocolate.

```
SELECT LOWER( 'chOCOLate' );
```

LTRIM function [String]

Removes leading blanks from the string.

Syntax

LTRIM(*string-expression* **)**

Parameters

- **string-expression** The string to be trimmed.

Returns

- VARCHAR
- NVARCHAR
- LONG VARCHAR
- LONG NVARCHAR

Remarks

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.

See also

- "RTRIM function [String]" on page 341
- "TRIM function [String]" on page 358
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

  The TRIM specifications defined by the SQL/2008 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' );
```

MAX function [Aggregate]

Returns the maximum *expression* value found in each group of rows.

Syntax 1

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.

For simple comparisons of two expressions, you can also use the GREATER function. See "GREATER function [Miscellaneous]" [*SQL Anywhere Server - SQL Reference*].

See also

- "MIN function [Aggregate]" on page 321

Standards and compatibility

- **SQL/2008** Core feature.

Example

The following statement returns the value 138948.000, representing the maximum salary in the Employees table.

```
SELECT MAX( Salary )
FROM Employees;
```

MIN function [Aggregate]

Returns the minimum expression value found in each group of rows.

Syntax 1

MIN([**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.

For simple comparisons of two expressions, you can also use the LESSER function. See "LESSER function [Miscellaneous]" [*SQL Anywhere Server - SQL Reference*].

See also

- "MAX function [Aggregate]" on page 320

Standards and compatibility

- **SQL/2008** Core feature.

Example

The following statement returns the value 24903.000, representing the minimum salary in the Employees table.

```
SELECT MIN( Salary )
FROM Employees;
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 22.

```
SELECT MINUTE( '1998-07-13 12:22:34' );
```

MINUTES function [Date and time]

The MINUTES function manipulates a TIMESTAMP, or returns the number of minutes between two TIMESTAMP values. See the Remarks section below.

Syntax 1

MINUTES(*timestamp-expression* **)**

Syntax 2

MINUTES(*timestamp-expression*, *timestamp-expression* **)**

Syntax 3

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 with Syntax 1 or Syntax 2.

TIME or TIMESTAMP with Syntax 3.

Remarks

The result of the MINUTES function depends on its arguments.

- **Syntax 1** If you pass a single *timestamp-expression* to the MINUTES function, it will return the number of minutes between midnight 0000-02-29 and *timestamp-expression* as an INTEGER.

  Note
  0000-02-29 is not meant to imply an actual date; it is the default date used by the MINUTES function.

- **Syntax 2** If you pass two TIMESTAMP values to the MINUTES function, the function returns the integer number of minutes between them.

- **Syntax 3** 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. Syntax 3 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 Syntax 1 is used with TIMESTAMP values greater than or equal to 4083-03-23 02:08:00.

Instead of Syntax 2, use the DATEDIFF function. Instead of Syntax 3, use the DATEADD function.

See also

- "DATEDIFF function [Date and time]" [*SQL Anywhere Server - SQL Reference*]
- "DATEADD function [Date and time]" [*SQL Anywhere Server - SQL Reference*]
- "CAST function [Data type conversion]" on page 279

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statements return the value 240, signifying that the second TIMESTAMP value is 240 minutes after the first. It is recommended that you use the second example (DATEDIFF).

```
SELECT MINUTES( '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' );

SELECT 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. Note that 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' );
```

ML_GET_SERVER_NOTIFICATION [System]

This function allows UltraLite users to use light weight polling to query a notifier on a MobiLink server for server-initiated sync requests. Not supported for UltraLiteJ

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}
  ```

  for example.
- **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.

See also

- "SYNC_PROFILE_OPTION_VALUE function [System]" on page 355

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

```
SELECT ML_GET_SERVER_NOTIFICATION('Notifier1',
'tcpip{host=sybase;port=1234}', 'MyKey')
```

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.

See also

- "REMAINDER function [Numeric]" on page 336

Standards and compatibility

- **SQL/2008** The MOD function is part of optional SQL/2008 language feature T441.

Example

The following statement returns the value 2.

```
SELECT MOD( 5, 3 );
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 7.

```
SELECT MONTH( '1998-07-13' );
```

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.

See also

- "DATEPART function [Date and time]" on page 295

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value September.

```
SELECT MONTHNAME( '1998-09-05' );
```

MONTHS function [Date and time]

The MONTHS function manipulates a TIMESTAMP, or returns the number of months between two TIMESTAMP values. See the Remarks section below.

Syntax 1

MONTHS(*timestamp-expression* **)**

Syntax 2

MONTHS(*timestamp-expression*, *timestamp-expression* **)**

Syntax 3

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.

  For information about casting data types, see "CAST function [Data type conversion]" on page 279.

Returns

INTEGER with Syntax 1 or Syntax 2.

TIMESTAMP with Syntax 3.

Remarks

The result of the MONTHS function depends on its arguments. The MONTHS function ignores hours, minutes, and seconds in its arguments.

- **Syntax 1** If you pass a single *timestamp-expression* to the MONTHS function, it will return the number of months between 0000-02 and *timestamp-expression* as an INTEGER.

  Note
  0000-02 is not meant to imply an actual date; it is the default date used by the MONTHS function.

- **Syntax 2** If you pass two TIMESTAMP values to the MONTHS function, the function returns the integer number of months between them.
- **Syntax 3** 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*.

Instead of Syntax 2, use the DATEDIFF function. Instead of Syntax 3, use the DATEADD function.

The value of MONTHS is calculated from the number of first days of the month between the two dates.

See also

- "DATEDIFF function [Date and time]" on page 292
- "DATEADD function [Date and time]" on page 291

Standards and compatibility

- **SQL/2008** Vendor extension.

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' );
```

NEWID function [Miscellaneous]

Generates a UUID (Universally Unique Identifier) value. A UUID is the same as a GUID (Globally Unique Identifier).

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.

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.

See also

- "The NEWID default" [*SQL Anywhere Server - SQL Usage*]
- "STRTOUUID function [String]" on page 350
- "UUIDTOSTR function [String]" on page 361

Standards and compatibility

- **SQL/2008** Vendor extension.

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 NEWID();
```

For example, the value returned might be 96603324-6FF6-49DE-BF7D-F44C1C7E6856.

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.

See also

- "GETDATE function [Date and time]" on page 304
- "CURRENT TIMESTAMP special value" on page 228

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the current date and time.

```
SELECT NOW( * );
```

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.

See also

- "CASE expressions" on page 249

Standards and compatibility

- **SQL/2008** Core feature.

Example

The following statement returns the value a:

```
SELECT NULLIF( 'a', 'b' );
```

The following statement returns NULL.

```
SELECT NULLIF( 'a', 'a' );
```

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 not.

  The pattern for UltraLite uses the following wildcards:

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).

See also

- “LOCATE function [String]” on page 316
- “UltraLite string functions” on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

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. Note that 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]%', 'SQLAnywhere12 has many new features' );
```

To get the first alphanumeric word in a string, you can use something like the following:

```
SELECT LEFT( @string, PATINDEX( '%[^a-zA-Z0-9]%', @string ) );
```

PI function [Numeric]

Returns the numeric value PI.

Syntax

PI([*****] **)**

Returns

DOUBLE

Standards and compatibility

- **SQL/2008** Vendor extension.

Remarks

This function returns a DOUBLE value.

PI(*) and PI() are semantically equivalent.

Example

The following statement returns the value 3.141592653...

```
SELECT PI( * );
```

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 and compatibility

- **SQL/2008** The POWER function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns the value 64.

```
SELECT POWER( 2, 6 );
```

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.

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 2.

```
SELECT QUARTER( '1987/05/02' );
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns a value of approximately 0.5236.

```
SELECT RADIANS( 30 );
```


RAND function [Numeric]

Returns a random number in the interval 0 to 1, with an optional seed. Not supported by UltraLiteJ

Syntax

RAND([*integer-expression*] **)**

Parameters

- **integer-expression** An optional seed used to create a random number. This argument allows you to create repeatable random number sequences.

Returns

DOUBLE

Remarks

The RAND function is a multiplicative linear congruential random number generator. See Park and Miller (1988), CACM 31(10), pp. 1192-1201 and Press et al. (1992), Numerical Recipes in C (2nd edition, Chapter 7, pp. 279). The result of calling the RAND function is a pseudo-random number *n* where $0 < n < 1$ (neither 0.0 nor 1.0 can be the result).

When a connection is made to the server, the random number generator seeds an initial value. Each connection is uniquely seeded so that it sees a different random sequence from other connections. You can also specify a seed value (*integer-expression*) as an argument. Normally, you should only do this once before requesting a sequence of random numbers through successive calls to the RAND function. If you initialize the seed value more than once, the sequence is restarted. If you specify the same seed value, the same sequence is generated. Seed values that are close in value generate similar initial sequences, with divergence further out in the sequence.

Never combine the sequence generated from one seed value with the sequence generated from a second seed value, in an attempt to obtain statistically random results. In other words, do not reset the seed value at any time during the generation of a sequence of random values.

The RAND function is treated as a non-deterministic function. The query optimizer does not cache the results of the RAND function.

For more information about non-deterministic functions, see "Function caching" [*SQL Anywhere Server - SQL Usage*].

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statements produce eleven random results. Each subsequent call to the RAND function where a seed is not specified continues to produce different results:

```
SELECT RAND( 1 );
SELECT RAND( ), RAND( ), RAND( ), RAND( ), RAND( );
SELECT RAND( ), RAND( ), RAND( ), RAND( ), RAND( );
```

The following example produces two sets of results with identical sequences, since the seed value is specified twice:

```
SELECT RAND( 1 ), RAND( ), RAND( ), RAND( ), RAND( );
SELECT RAND( 1 ), RAND( ), RAND( ), RAND( ), RAND( );
```

The following example produces five results that are near each other in value, and do not have a random distribution. For this reason, calling the RAND function more than once with similar seed values is not recommended:

```
SELECT RAND( 1 ), RAND( 2 ), RAND( 3 ), RAND( 4 ), RAND( 5 );
```

The following example produces five identical results, and should be avoided:

```
SELECT RAND( 1 ), RAND( 1 ), RAND( 1 ), RAND( 1 ), RAND( 1 );
```

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.

See also

- "MOD function [Numeric]" on page 325

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 2.

```
SELECT REMAINDER( 5, 3 );
```


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

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.

See also

- "REPLICATE function [String]" on page 338
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value repeatrepeatrepeat.

```
SELECT REPEAT( 'repeat', 3 );
```

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 *replacement-string* is an empty string, all occurrences of *search-string* are deleted.

Returns

- LONG VARCHAR
- LONG NVARCHAR

Remarks

This function replaces all occurrences.

Comparisons are case-sensitive on case-sensitive databases.

See also

- "SUBSTRING function [String]" on page 352
- "CHARINDEX function [String]" on page 283
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

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%';
```

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

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.

See also

- "REPEAT function [String]" on page 337
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value repeatrepeatrepeat.

```
SELECT REPLICATE( 'repeat', 3 );
```

RIGHT function [String]

Returns the rightmost characters of a string.

Syntax

RIGHT(*string-expression*, *integer-expression* **)**

Parameters

- **string-expression** The string to be left-truncated.
- **integer-expression** The number of characters at the end of the string to return.

Returns

- LONG VARCHAR
- LONG NVARCHAR

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 column. 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.

See also

- "LEFT function [String]" on page 313
- "International languages and character sets" [*SQL Anywhere Server - Database Administration*]
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the last 5 characters of each Surname value in the Customers table.

```
SELECT RIGHT( Surname, 5) FROM Customers;
```

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.

See also

- "TRUNCNUM function [Numeric]" on page 359

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 123.200.

```
SELECT ROUND( 123.234, 1 );
```

RTRIM function [String]

Removes trailing blanks from the string.

Syntax

RTRIM(*string-expression* **)**

Parameters

- **string-expression** The string to be trimmed.

Returns

- VARCHAR
- NVARCHAR
- LONG VARCHAR
- LONG NVARCHAR

Remarks

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.

See also

- “TRIM function [String]” on page 358
- “LTRIM function [String]” on page 319
- “UltraLite string functions” on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

  The TRIM specifications defined by the SQL/2008 standard (LEADING and TRAILING) are supplied by the SQL Anywhere LTRIM and RTRIM functions respectively.

Example

The following statement returns the string Test Message, with all trailing blanks removed.

```
SELECT RTRIM( 'Test Message     ' );
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 25.

```
SELECT SECOND( '1998-07-13 21:21:25' );
```

SECONDS function [Date and time]

The SECONDS function manipulates a TIMESTAMP, or returns the number of seconds between two TIMESTAMP values. See the Remarks section below.

Syntax 1

SECONDS(*timestamp-expression* **)**

Syntax 2

SECONDS(*timestamp-expression*, *timestamp-expression* **)**

Syntax 3

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 with Syntax 1.

SIGNED BIGINT with Syntax 2.

TIME or TIMESTAMP with Syntax 3.

Remarks

The result of the SECONDS function depends on its arguments.

- **Syntax 1** If you pass a single *timestamp-expression* to the SECONDS function, it will return the number of seconds between midnight 0000-02-29 and *timestamp-expression* as an UNSIGNED BIGINT.

  Note
  0000-02 is not meant to imply an actual date; it is the default date used by the SECONDS function.

- **Syntax 2** If you pass two TIMESTAMP values to the SECONDS function, the function returns the integer number of seconds between them as a SIGNED BIGINT value.

- **Syntax 3** 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.

Instead of Syntax 2, use the DATEDIFF function. Instead of Syntax 3, use the DATEADD function.

See also

- “CAST function [Data type conversion]” on page 279
- “DATEADD function [Date and time]” on page 291
- “DATEDIFF function [Date and time]” on page 292

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statements return the value 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' );
SELECT 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' );
```

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* **)**

Remarks

For some queries, the execution plan for UltraLite may differ from the plan selected for SQL Anywhere.

Parameters

- **string-expression** The SQL statement, which is commonly a SELECT statement, but can also be an UPDATE or DELETE statement.

Returns

LONG VARCHAR

See also

- "EXPLANATION function [Miscellaneous]" on page 303

Standards and compatibility

- **SQL/2008** Vendor extension.

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 Departments WHERE DepartmentID > 100' );
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value -1

```
SELECT SIGN( -550 );
```

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

SMALL INT

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.

See also

- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 75, indicating that the two values are 75% similar.

```
SELECT SIMILAR( 'toast', 'coast' );
```

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.

See also

- "ASIN function [Numeric]" on page 275
- "COS function [Numeric]" on page 287
- "COT function [Numeric]" on page 288
- "TAN function [Numeric]" on page 356

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the SIN value of 0.52.


```
SELECT SIN( 0.52 );
```

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.

See also

- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns two identical numbers, 3827, representing the sound of each name.

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' );
```

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.

See also

- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns a string containing 10 spaces.

```
SELECT SPACE( 10 );
```

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 and compatibility

- **SQL/2008** The SQRT function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns the value 3.

```
SELECT SQRT( 9 );
```


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 );
```

Note
The maximum length that is supported is 128. Any length that is not between 1 and 128 yields a result of NULL.

See also

- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

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 );
```

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 more than one argument is supplied, they are concatenated into a single string.

Returns

- LONG VARCHAR
- LONG NVARCHAR
- LONG BINARY

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.

See also

- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value *testing123*.

```
SELECT STRING( 'testing', NULL, 123 );
```

STRTOUUID function [String]

Converts a string value to a unique identifier (UUID or GUID) value.

Not needed in newer databases
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.

For more information, see "Data types in UltraLite" on page 230.

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.

See also

- "UUIDTOSTR function [String]" on page 361
- "NEWID function [Miscellaneous]" on page 328
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

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 NVARCHAR

See also

- "INSERTSTR function [String]" on page 309
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value chocolate pie.

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' );
```

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

Remarks

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/2008 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.

If *string-expression* is of binary data type, the SUBSTRING function behaves as BYTE_SUBSTR.

To obtain characters at the end of a string, use the RIGHT function.

Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics.

See also

- "BYTE_SUBSTR function [String]" on page 279
- "LEFT function [String]" on page 313
- "RIGHT function [String]" on page 339
- "CHARINDEX function [String]" on page 283
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** SUBSTRING is a core feature of the SQL/2008 standard. The standard's implementation differs slightly from the SQL Anywhere implementation: in the standard, SUBSTRING is defined with three parameters using the keywords FROM and FOR, neither of which are required by SQL Anywhere.

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
SUBSTR('abcdefgh', -2, 2)	Returns an empty string

SUM function [Aggregate]

Returns the total of the specified expression for each group of rows.

Syntax 1

SUM([**DISTINCT**] *expression* **)**

Parameters

- **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. See "CAST function [Data type conversion]" [*SQL Anywhere Server - SQL Reference*].

See also

- "COUNT function [Aggregate]" on page 288
- "AVG function [Aggregate]" on page 277

Standards and compatibility

- **SQL/2008** Core feature.

Example

The following statement returns the value 3749146.740.

```
SELECT SUM( Salary )
FROM Employees;
```

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

See also

- "TIMESTAMP WITH TIME ZONE data type" [*SQL Anywhere Server - SQL Reference*]
- "SYSDATETIMEOFFSET function [Date and time]" [*SQL Anywhere Server - SQL Reference*]

Standards and compatibility

- **SQL/2008** Vendor extension.

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;
```

SYNC_PROFILE_OPTION_VALUE function [System]

Returns the value of the option corresponding to the given option name. Not supported for UltraLiteJ.

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.

See also

- "ML_GET_SERVER_NOTIFICATION [System]" on page 324

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

Consider the profile:

```
MobiLinkUid=joe;Stream=tcpip{host=sybase;port=1234};Ping=1
```

- **MobiLinkUid** joe

- **Stream** tcpip{host=sybase;port=1234}
- **Stream.host** sybase
- **Stream.port** 1234
- **Ping** 1

SYNC_PROFILE_PARM function [System]

Returns the description of an UltraLite sync profile. Not supported for UltraLiteJ.

Syntax

SYNC_PROFILE_PARM(*profile_name*, *parameter-number***)**

Parameters

- **profile_name** The name of the sync profile to inspect.
- **parameter number** The number of the parameter to retrieve the corresponding value for.

Returns

The parameters are returned in the form parm=value. If parameter 0 is requested, the entire synchronization profile is returned (truncated at 2000 characters).

Standards and compatibility

- **SQL/2008** Vendor extension.

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.

See also

- "COS function [Numeric]" on page 287
- "SIN function [Numeric]" on page 346

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value of the tan of 0.52.

```
SELECT TAN( 0.52 );
```

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

See also

- "TIMESTAMP WITH TIME ZONE data type" [*SQL Anywhere Server - SQL Reference*]

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following example changes a time zone offset value from -4.00 hours to +11.00 hours.

```
SELECT TODATETIMEOFFSET ('2009-04-03 14:45:12.123-04:00','+11:00');
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following statements return the current day according to the system clock.

```
SELECT TODAY( * );
SELECT CURRENT DATE;
```

TRIM function [String]

Removes leading and trailing blanks from a string.

Syntax

TRIM(*string-expression* **)**

Parameters

- **string-expression** The string to be trimmed.

Returns

- VARCHAR
- NVARCHAR
- LONG VARCHAR
- LONG NVARCHAR

See also

- “LTRIM function [String]” on page 319
- “RTRIM function [String]” on page 341
- “UltraLite string functions” on page 271

Standards and compatibility

- **SQL/2008** The TRIM function is a SQL/2008 core feature.

SQL Anywhere does not support the additional parameters *trim specification* and *trim character*, as defined in SQL/2008. The SQL Anywhere implementation of TRIM corresponds to a TRIM specification of BOTH.

For the other TRIM specifications defined by the SQL/2008 standard (LEADING and TRAILING), SQL Anywhere supplies the LTRIM and RTRIM functions respectively.

Example

The following statement returns the value chocolate with no leading or trailing blanks.

```
SELECT TRIM( '   chocolate   ' );
```

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

If any parameter is NULL, the result is NULL.

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.

See also

- "ROUND function [Numeric]" on page 340

Standards and compatibility

- **SQL/2008** Vendor extension.

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 );
```

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

CHAR, VARCHAR, LONG VARCHAR, NCHAR, NVARCHAR, or LONG NVARCHAR corresponding on the type of the argument.

Remarks

This function is identical to the UPPER function.

See also

- “UPPER function [String]” on page 360
- “LCASE function [String]” on page 312
- “UltraLite string functions” on page 271

Standards and compatibility

- **SQL/2008** Vendor extension. The UPPER function is SQL/2008 compliant.

Example

The following statement returns the value CHOCOLATE.

```
SELECT UCASE( 'ChocoLate' );
```

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

CHAR, VARCHAR, LONG VARCHAR, NCHAR, NVARCHAR, or LONG NVARCHAR corresponding to the data type of the argument.

Remarks

This function is identical to the UCASE function.

See also

- "UCASE function [String]" on page 360
- "LCASE function [String]" on page 312
- "LOWER function [String]" on page 318
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** The UPPER function is a core feature of the SQL/2008 standard.

Example

The following statement returns the value CHOCOLATE.

```
SELECT UPPER( 'ChocoLate' );
```

UUIDTOSTR function [String]

Converts a unique identifier value (UUID, also known as GUID) to a string value.

Not needed in newer databases
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.

For more information, see "Data types in UltraLite" on page 230.

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 if you want to view a UUID value.

See also

- "NEWID function [Miscellaneous]" on page 328
- "STRTOUUID function [String]" on page 350
- "UltraLite string functions" on page 271

Standards and compatibility

- **SQL/2008** Vendor extension.

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 command:

```
SELECT UUIDTOSTR(pk), c1 FROM mytab;
```

The UUIDTOSTR function is not needed for databases created with version 9.0.2 or later.

WEEKS function [Date and time]

The WEEKS function manipulates a TIMESTAMP, or returns the number of weeks between two TIMESTAMP values. See the Remarks section below.

Syntax 1

WEEKS(*timestamp-expression* **)**

Syntax 2

WEEKS(*timestamp-expression*, *timestamp-expression* **)**

Syntax 3

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 with Syntax 1 or Syntax 2.

TIMESTAMP with Syntax 3.

Remarks

Given a single date (Syntax 1), the WEEKS function returns the number of weeks since 0000-02-29.

Given two dates (Syntax 2), 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.

For more information about the DATEDIFF function, see "DATEDIFF function [Date and time]" on page 292.

Given a date and an integer (Syntax 3), the WEEKS function adds the integer number of weeks to *timestamp-expression*. With Syntax 3, 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. Instead of Syntax 3, use the DATEADD function.

For more information about the DATEADD function, see "DATEADD function [Date and time]" on page 291.

See also

For information about casting data types, see "CAST function [Data type conversion]" on page 279.

Standards and compatibility

- **SQL/2008** Vendor extension.

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);
```

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 and compatibility

- **SQL/2008** Vendor extension.

Example

The following example returns the value 2001.

```
SELECT YEAR( '2001-09-12' );
```

YEARS function [Date and time]

The YEARS function manipulates a TIMESTAMP, or returns the number of years between two TIMESTAMP values. See the Remarks section below.

Syntax 1

YEARS(*timestamp-expression* **)**

Syntax 2

YEARS(*timestamp-expression*, *timestamp-expression* **)**

Syntax 2

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.

  For information about casting data types, see "CAST function [Data type conversion]" on page 279.

Returns

SMALLINT with Syntax 1 or Syntax 2.

TIMESTAMP with Syntax 3.

Remarks

The value of YEARS is computed by counting the number of first days of the year between the two dates.

See also

- "DATEDIFF function [Date and time]" on page 292
- "DATEADD function [Date and time]" on page 291

Standards and compatibility

- **SQL/2008** Vendor extension.

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' );
```

YMD function [Date and time]

Returns a date value corresponding to the given year, month, and day of the month. Arguments are SMALLINT 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 and compatibility

- **SQL/2008** Vendor extension.

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 );
```

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. See "Syntax conventions" [*SQL Anywhere Server - SQL Reference*].

- Using UltraLite SQL creates a transaction. A transaction consists of all changes (INSERTs, UPDATEs, and DELETEs) since the last ROLLBACK or COMMIT. See "UltraLite transaction processing" on page 14.

  These changes can be made permanent by executing a COMMIT. A ROLLBACK statement causes the changes to be removed. See "COMMIT statement [UltraLite] [UltraLiteJ]" on page 376 and "ROLLBACK statement [UltraLite] [UltraLiteJ]" on page 402.

- If you are looking for statements used by Interactive SQL, see "SQL statements" [*SQL Anywhere Server - SQL Reference*]. Statements used by Interactive SQL have **[Interactive SQL]** after the statement name. For example, "CONFIGURE statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*].

UltraLite statement categories

SQL statements are organized and identified by the initial word in a 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. See "SELECT statement [UltraLite] [UltraLiteJ]" on page 402.

- **Data manipulation statements** Allow you to change content in the database. Data manipulation is achieved with:

  - "DELETE statement [UltraLite] [UltraLiteJ]" on page 391
  - "INSERT statement [UltraLite] [UltraLiteJ]" on page 397
  - "UPDATE statement [UltraLite] [UltraLiteJ]" on page 410

- **Data definition statements** Allow you to define the structure or schema of a database. The schema can be changed with:

  - "ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]" on page 368
  - "CREATE INDEX statement [UltraLite] [UltraLiteJ]" on page 376
  - "CREATE TABLE statement [UltraLite] [UltraLiteJ]" on page 384
  - "DROP INDEX statement [UltraLite] [UltraLiteJ]" on page 392
  - "DROP TABLE statement [UltraLite] [UltraLiteJ]" on page 394
  - "ALTER TABLE statement [UltraLite] [UltraLiteJ]" on page 371
  - "TRUNCATE TABLE statement [UltraLite] [UltraLiteJ]" on page 408

- **Transaction control statements** Allow you to control transactions within your UltraLite application. Transaction control is achieved with:
  - "CHECKPOINT statement [UltraLite]" on page 375
  - "COMMIT statement [UltraLite] [UltraLiteJ]" on page 376
  - "ROLLBACK statement [UltraLite] [UltraLiteJ]" on page 402
- **Synchronization management** Allow you to temporarily control synchronization with a MobiLink server. Synchronization management is achieved with:
  - "START SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 405
  - "STOP SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 406
  - "CREATE PUBLICATION statement [UltraLite] [UltraLiteJ]" on page 378
  - "ALTER PUBLICATION statement [UltraLite][UltraLiteJ]" on page 369
  - "DROP PUBLICATION statement [UltraLite] [UltraLiteJ]" on page 392

See also

- "Expressions in UltraLite" on page 246
- "Operators in UltraLite" on page 259

ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]

Use this statement to modify 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). See "UltraLite Initialize Database utility (ulinit)" on page 197, and "UltraLite Database Unload utility (ulunload)" on page 214.

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

SQLE_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.

See also

- "Deploying UltraLite schema upgrades" on page 51

Example

The following statement modifies the schema of the database using a SQL script, *MySchema.sql*:

```
ALTER DATABASE SCHEMA FROM FILE 'MySchema.sql';
```

ALTER PUBLICATION statement [UltraLite][UltraLiteJ]

Use this statement to alter a publication. A publication identifies data in a remote database that is to be synchronized.

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*

Side effects

Automatic commit.

See also

- "Search conditions in UltraLite" on page 253
- "Designing synchronization in UltraLite" on page 99
- "CREATE PUBLICATION statement [UltraLite] [UltraLiteJ]" on page 378
- "DROP PUBLICATION statement [UltraLite] [UltraLiteJ]" on page 392
- "START SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 405
- "STOP SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 406

Example

The following ALTER PUBLICATION statement adds the Customers table to the pub_contact publication.

```
ALTER PUBLICATION pub_contact
   ADD TABLE Customers;
```

ALTER SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]

Use this statement to alter an UltraLite or UltraLiteJ synchronization profile. Synchronization profiles define how an UltraLite or UltraLiteJ databases synchronize with the MobiLink server.

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

See "CREATE SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 379 for a list of the synchronization profile options supported by UltraLite and UltraLiteJ.

The REPLACE clause supported by earlier versions of UltraLite has been deprecated. Use **CREATE SYNCHRONIZATION PROFILE** with the **OR REPLACE** clause instead. See "ALTER SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 369.

Side effects

None.

See also

- "DROP SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 393
- "SYNCHRONIZE statement [UltraLite] [UltraLiteJ]" on page 407

Example

The following is an example of the ALTER SYNCHRONIZATION PROFILE...REPLACE statement:

```
CREATE SYNCHRONIZATION PROFILE myProfile1;
ALTER SYNCHRONIZATION PROFILE myProfile1
   REPLACE 'publication=p1;uploadonly=on';
```

The following is an example of the ALTER SYNCHRONIZATION PROFILE...MERGE statement.


```
CREATE SYNCHRONIZATION PROFILE myProfile2 'publication=p1;verbosity=high';
ALTER SYNCHRONIZATION PROFILE myProfile2
   MERGE'publication=p2;uploadonly=on';
```

ALTER TABLE statement [UltraLite] [UltraLiteJ]

Use this statement to modify 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**

Parameters

add-clause 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. For descriptions of the keywords and subclauses for this clause, see "CREATE TABLE statement [UltraLite] [UltraLiteJ]" on page 384.

- **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. For descriptions of the keywords and subclauses for this clause, see "CREATE TABLE statement [UltraLite] [UltraLiteJ]" on page 384.

> **Note**
> You cannot add a primary key in UltraLite or UltraLiteJ.

modify-clause Change a single column definition. Note that 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. For a full explanation of *column-definition*, see "CREATE TABLE statement [UltraLite] [UltraLiteJ]" on page 384.

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. For a full explanation of *table-constraint*, see "CREATE TABLE statement [UltraLite] [UltraLiteJ]" on page 384.

> **Note**
> You cannot drop a primary key in UltraLite or UltraLiteJ.

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*. Note that 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*. Note that 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*. Note that any applications using the old constraint name need to be modified.

> **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.

- **SYNCHRONIZE** Determines whether a table can be synchronized or not and whether all rows are uploaded or just changes to the table are uploaded. Valid values are ON (default setting - the table can be synchronized and only changes to the table are sent in the upload), OFF (the table cannot be synchronized and it is an error to include the table in a publication), and ALL (the table can be synchronized and all rows in the table are sent in the upload).

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.

A table whose name ends with **nosync** can only be renamed to a table name that also ends with **nosync**. See "Non-synchronizing tables in UltraLite" on page 101.

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 Dispose method for all data objects (for example, ULDataReader). See "Dispose method" [*UltraLite - .NET Programming*].

Statements are not released if schema changes are initiated at the same time. See "Schema changes with DDL statements" on page 10.

See also

- "CREATE TABLE statement [UltraLite] [UltraLiteJ]" on page 384
- "DROP TABLE statement [UltraLite] [UltraLiteJ]" on page 394
- "Data types in UltraLite" on page 230
- "Altering tables" [*SQL Anywhere Server - SQL Usage*]
- "Using table and column constraints" [*SQL Anywhere Server - SQL Usage*]
- "Overriding partition sizes for autoincremented columns" on page 98
- "Determining the most recently assigned GLOBAL AUTOINCREMENT value" on page 97

Examples

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
MODIFY Street CHAR(50);
```

ALTER USER statement [UltraLite]

Alters user settings. UltraLite only. Not supported for UltraLiteJ.

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

User IDs and passwords cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons

A password can be either a valid identifier, or a string (maximum 255 bytes) placed in single quotes. Passwords are case sensitive. It is recommended that the password be composed of 7-bit ASCII characters, as other characters may not work correctly if the database server cannot convert them from the client's character set to UTF-8.

Permissions

Any user can change their own password. All other changes require DBA authority.

Side effects

None.

See also

- "CREATE USER statement [UltraLite]" on page 390
- "DROP USER statement [UltraLite]" on page 394

Standards and compatibility

- **SQL/2003** Vendor extension.

Example

The following alters a user named SQLTester. The password is set to "welcome".

```
ALTER USER SQLTester IDENTIFIED BY welcome;
```

CHECKPOINT statement [UltraLite]

Use this statement to 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.

See also

- "Flushing single or grouped transactions" on page 89
- "COMMIT statement [UltraLite] [UltraLiteJ]" on page 376
- "UltraLite COMMIT_FLUSH connection parameter" on page 170
- UltraLite embedded SQL: "ULCheckpoint method" [*UltraLite - C and C++ Programming*]
- UltraLite C++: "Checkpoint method" [*UltraLite - C and C++ Programming*]

Example

The following statement performs a checkpoint of the database:

```
CHECKPOINT;
```

COMMIT statement [UltraLite] [UltraLiteJ]

Use this statement to make 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.

See also

- "CHECKPOINT statement [UltraLite]" on page 375
- "ROLLBACK statement [UltraLite] [UltraLiteJ]" on page 402

Example

The following statement makes the changes in the current transaction permanent in the database:

```
COMMIT;
```

CREATE INDEX statement [UltraLite] [UltraLiteJ]

Use this statement to create 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 will not be 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 property in effect for the database. To learn the default size, see "Accessing UltraLite database properties" on page 162. This is not supported for UltraLiteJ.

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.

UltraLite can also use execution plans to optimize queries. See "Execution plans in UltraLite" on page 262.

For UltraLite.NET users: You cannot execute this statement unless you also call the Dispose method for all data objects (for example, ULDataReader). See "Dispose method" [*UltraLite - .NET Programming*].

Statements are not released if schema changes are initiated at the same time. See "Schema changes with DDL statements" on page 10.

Side effects

- Automatic commit.

See also

- "UltraLite performance and optimization" on page 81
- "DROP INDEX statement [UltraLite] [UltraLiteJ]" on page 392
- "UltraLite max_hash_size creation parameter" on page 143
- "Working with UltraLite indexes" on page 61

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 );
```

CREATE PUBLICATION statement [UltraLite] [UltraLiteJ]

Use this statement to create a publication. A publication identifies synchronized data in an UltraLite remote database.

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. See "Search conditions in UltraLite" on page 253.

  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 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.

See also

- "UltraLite clients" on page 93
- "DROP PUBLICATION statement [UltraLite] [UltraLiteJ]" on page 392
- "ALTER PUBLICATION statement [UltraLite][UltraLiteJ]" on page 369
- "Search conditions in UltraLite" on page 253

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'
);
```

CREATE SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]

Use this statement to create or replace an UltraLite or UltraLiteJ synchronization profile. Synchronization profiles define how an UltraLite database synchronizes with the MobiLink server.

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

There are two possible ways to make changes to an existing synchronization profile. The first is to use the REPLACE clause. This will replace the contents of the synchronization profile with whatever is contained in the new sync-option string. This is the same as dropping the synchronization profile and then creating one with the same name but using the new string. Note, therefore, that a synchronization profile does not need to contain a full synchronization definition because parameters can be merged in (or overridden) at synchronization time.

The second way to modify a synchronization profile is to use the MERGE clause. When using this clause, only the sync options that are specified in the MERGE clause are changed. To remove a sync 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{}.

The following table lists the synchronization profile options supported by UltraLite and UltraLiteJ.

Synchronization profile option	Valid values	Description
AllowDownloadDupRows (UltraLite only)	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 synchronisation to fail. The default value is "no." See "Additional Parameters synchronization parameter" on page 111.
AuthParms (UltraLite and UltraLiteJ)	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. See "Authentication Parameters synchronization parameter" on page 112.

Synchronization profile option	Valid values	Description
CheckpointStore (UltraLite only)	Boolean	Adds additional checkpoints of the database during synchronization to limit database growth during the synchronization process. See "Additional Parameters synchronization parameter" on page 111.
ContinueDownload (UltraLite only)	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. See "Resuming failed downloads" [*MobiLink - Server Administration*].
DisableConcurrency (UltraLite only)	Boolean	Disallow database access from other threads during synchronization. See "Additional Parameters synchronization parameter" on page 111.
DownloadOnly (UltraLite and UltraLiteJ)	Boolean	Performs a download-only synchronization. See "Download Only synchronization parameter" on page 115.
KeepPartialDownload (UltraLite only)	Boolean	Controls whether UltraLite keeps a partial download if a communication error occurs. By default, UltraLite does not roll back partially downloaded changes. See "Keep Partial Download synchronization parameter" on page 117.
MobiLinkPwd (UltraLite and UltraLiteJ)	String	Specifies the existing MobiLink password associated with the user name. See "MobiLinkPwd (mp) extended option" [*MobiLink - Client Administration*].
MobiLinkUid (UltraLite and UltraLiteJ)	String	Specifies the MobiLink user name. See "-u dbmlsync option (deprecated)" [*MobiLink - Client Administration*]. See "-mn dbmlsync option" [*MobiLink - Client Administration*].

Synchronization profile option	Valid values	Description
NewMobiLinkPwd (UltraLite and UltraLiteJ)	String	Supplies a new password for the MobiLink user. Use this option when you want to change an existing password. See "-mn dbmlsync option" [*MobiLink - Client Administration*].
Ping (UltraLite and UltraLiteJ)	Boolean	Confirms communications with the server only; no synchronization is performed. See "Ping synchronization parameter" on page 121.
Publications (UltraLite and UltraLiteJ)	String (comma separated)	Specifies the publications(s) to synchronize. The publications determine the tables on the remote 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. See "Publications in UltraLite" on page 102.
ScriptVersion (UltraLite and UltraLiteJ)	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. See "ScriptVersion (sv) extended option" [*MobiLink - Client Administration*].
SendColumnNames (UltraLite and UltraLiteJ)	String	Specifies that column names should be sent to the MobiLink server as part of the upload file when synchronizing. By default, column names are not sent. See "Send Column Names synchronization parameter" on page 124.
SendDownloadACK (UltraLite and UltraLiteJ)	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. See "Send Download Acknowledgement synchronization parameter" on page 125.
Stream (UltraLite and UltraLiteJ)	String (with sublist)	Specifies the MobiLink network synchronization protocol. See "Stream Type synchronization parameter" on page 127.

Synchronization profile option	Valid values	Description
TableOrder (UltraLite and UltraLiteJ)	String (comma separated)	Specifies the order of tables in the upload. By default, UltraLite and UltraLiteJ select an order based on foreign key relationships. See "Additional Parameters synchronization parameter" on page 111.
UploadOnly (UltraLite and UltraLiteJ)	String	Specifies that synchronization will only include an upload, and no download will occur. See "Upload Only synchronization parameter" on page 130.

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.

Side effects

None.

See also

- "ALTER SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 369
- "DROP SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 393
- "SYNCHRONIZE statement [UltraLite] [UltraLiteJ]" on page 407

Example

The following creates a synchronization profile called Test1.

```
CREATE SYNCHRONIZATION PROFILE Test1
'MobiLinkUid=mary;Stream=TCPIP{host=192.168.1.1;port=1234}'
```

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}'`.

CREATE TABLE statement [UltraLite] [UltraLiteJ]

Use this statement to create 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*) [**CASCADE DELETE**]
[*column-constraint*]

column-default :
AUTOFILENAME(*prefix*,*extension*)
| **GLOBAL AUTOINCREMENT** [**(** *number* **)**]
| **AUTOINCREMENT**
| **CURRENT DATE**
| **CURRENT TIME**
| **CURRENT 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**

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. See "Identifiers in UltraLite" on page 225.

  UltraLiteJ, using BlackBerry OS 4.2 or J2SE, supports the partitioning of database files such that external files may now be used to store large BLOB values, with the files referenced using the **file_name** and **file_contents** columns. The **file_name** column stores a data type of **CHAR**(*size*) and the **file_contents** column stores a **LONG BINARY** data type. This column is read-only.

  See "Data types in UltraLite" on page 230.

- **data-type** The data type of the column. See "Data types in UltraLite" on page 230.

- **[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 UltraLiteJ database.

    The **file_name** column 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. The filename must be a valid filename. For the BlackBerry, relative filenames are resolved against the database option **OPTION_BLOB_FILE_BASE_DIR**. If UltraLite determines that a filename does not begin with the prefix "*file://*", it will prepend the filename with the value of the

OPTION_BLOB_FILE_BASE_DIR option before attempting to open it. See "OPTION_BLOB_FILE_BASE_DIR variable" [*UltraLiteJ*].

- **AUTOINCREMENT** UltraLiteJ only.

  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.

  Tip
  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.

  Tip
  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.

  See "Using GLOBAL AUTOINCREMENT in UltraLite" on page 95, and "UltraLite global_database_id option" on page 165.

- **[NOT] NULL** Controls whether the column can contain NULLs.

- **NEWID( )** A function that generates a unique identifier value. See "NEWID function [Miscellaneous]" on page 328.

- **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. See "CURRENT TIMESTAMP special value" on page 228.

- **CURRENT DATE** Stores the current year, month, and day. See "CURRENT DATE special value" on page 227.

- **CURRENT TIME** Stores the current hour, minute, second and fraction of a second. See "CURRENT TIME special value" on page 228.

- **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*) [CASCADE DELETE]** UltraLiteJ only.

  The **file_contents** column must specify an existing **CHAR** column as the **STORE AS FILE** argument. The **file_contents** column behaves as a read-only column.

  On deletion of a row containing a BLOB file, the file will be deleted if **CASCADE DELETE** is specified. Otherwise the file will be left in the file system and will no longer be a property of the database. Specifying the **CASCADE DELETE** clause ensures that a file is never inserted into the database twice. On rollback of an insert, the file will not be deleted even if **CASCADE DELETE** is specified.

  An update that changes the contents of a filename column is identical to a **DELETE** followed by an **INSERT**. If **CASCADE DELETE** is specified, the file pointed to by the old filename will be deleted. If a row with a BLOB file is updated and the filename column is not changed, or if the filename column is updated to be the exact same value, the file is protected from being deleted. A BLOB file column may not be changed in an **UPDATE** statement.

- **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** UltraLite only. Not supported for UltraLiteJ. 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 property in effect for the database. To learn the default size, see "Accessing UltraLite database properties" on page 162. This is not supported for UltraLiteJ.

sync-constraint clause Specify a sync constraint to determine whether a table can be synchronized or not.

- **SYNCHRONIZE** Determines whether a table can be synchronized or not and whether all rows are uploaded or just changes to the table are uploaded. Valid values are ON (default setting - the table can be synchronized and only changes to the table are sent in the upload), OFF (the table cannot be synchronized and it is an error to include the table in a publication), and ALL (used for UltraLite only - the table can be synchronized and all rows in the table are sent in the upload).

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 schema changes are initiated at the same time. See "Schema changes with DDL statements" on page 10.

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 Dispose method for all data objects (for example, ULDataReader). See "Dispose method" [*UltraLite - .NET Programming*].

Synchronization of external BLOB columns (UltraLiteJ only) On the consolidated database, the filename column is stored as a regular **CHAR** column and the BLOB file column is stored as a regular BLOB (**LONG BINARY**) column. On a download, the filename column is ignored and a new filename is generated using the database option (Connection.OPTION_BLOB_FILE_BASE_DIR) and the prefix and extension strings specified on the **DEFAULT AUTOFILENAME** clause. For J2SE the syntax is `<database_option_blob_file_base_dir><prefix><auto generated integer value>.<extension>` and for the BlackBerry the syntax is `<prefix><auto generated integer value>.<extension>`. Therefore, for the BlackBerry generated filenames are always relative.

Side effects

Automatic commit.

See also

- "Expressions in UltraLite" on page 246
- "DROP TABLE statement [UltraLite] [UltraLiteJ]" on page 394
- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "Data types in UltraLite" on page 230
- "Overriding partition sizes for autoincremented columns" on page 98

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 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.

```
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)
);
```

CREATE USER statement [UltraLite]

Creates a database user or group. UltraLite only. Not supported for UltraLiteJ.

Syntax

CREATE USER *user-name* [**IDENTIFIED BY** *password*]

Parameters

user-name The name of the user you are creating.

Remarks

You do not have to specify a password for the user. A user without a password cannot connect to the database. This is useful if you are creating a group and do not want anyone to connect to the database using the group user ID. A user ID must be a valid identifier.

User IDs and passwords cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons

A password can be either a valid identifier, or a string (maximum 255 bytes) placed in single quotes. Passwords are case sensitive. It is recommended that the password be composed of 7-bit ASCII characters, as other characters may not work correctly if the database server cannot convert them from the client's character set to UTF-8.

Permissions

DBA authority

Side effects

None.

See also

- "ALTER USER statement [UltraLite]" on page 374
- "DROP USER statement [UltraLite]" on page 394

Standards and compatibility

- **SQL/2003** Vendor extension.

Example

The following example creates a user named SQLTester with the password welcome.

```
CREATE USER SQLTester IDENTIFIED BY welcome;
```

DELETE statement [UltraLite] [UltraLiteJ]

Use this statement to delete 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. See "Search conditions in UltraLite" on page 253.
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. See "UltraLite row states" on page 12.

See also

- "START SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 405
- "STOP SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 406

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;
```

DROP INDEX statement [UltraLite] [UltraLiteJ]

Use this statement to delete 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 Dispose method for all data objects (for example, ULDataReader). See "Dispose method" [*UltraLite - .NET Programming*].

Statements are not released if schema changes are initiated at the same time. See "Schema changes with DDL statements" on page 10.

See also

- "CREATE INDEX statement [UltraLite] [UltraLiteJ]" on page 376
- "Working with UltraLite indexes" on page 61

Example

The following statement deletes a fictitious index, fin_codes_idx, on the FinancialData table:

```
DROP INDEX FinancialData.fin_codes_idx;
```

DROP PUBLICATION statement [UltraLite] [UltraLiteJ]

Use this statement to delete 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.

See also

- "Designing synchronization in UltraLite" on page 99
- "ALTER PUBLICATION statement [UltraLite][UltraLiteJ]" on page 369
- "CREATE PUBLICATION statement [UltraLite] [UltraLiteJ]" on page 378

Example

The following statement drops the pub_contact publication.

```
DROP PUBLICATION pub_contact;
```

DROP SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]

Use this statement to delete an UltraLite or UltraLiteJ synchronization profile. Synchronization profiles define how an UltraLite or UltraLiteJ database synchronizes with the MobiLink server.

Syntax

DROP SYNCHRONIZATION PROFILE [**IF EXISTS**] *sync-profile-name*

Parameters

- **sync-profile-name** The name of the synchronization profile.

Remarks

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.

See also

- "CREATE SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 379
- "ALTER SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 369
- "SYNCHRONIZE statement [UltraLite] [UltraLiteJ]" on page 407

Example

The following example shows the syntax for dropping a synchronization profile called Test1.

```
DROP SYNCHRONIZATION PROFILE Test1;
```

DROP TABLE statement [UltraLite] [UltraLiteJ]

Use this statement to remove 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 Dispose method for all data objects (for example, ULDataReader). See "Dispose method" [*UltraLite - .NET Programming*].

Statements are not released if schema changes are initiated at the same time. See "Schema changes with DDL statements" on page 10.

See also

- "ALTER TABLE statement [UltraLite] [UltraLiteJ]" on page 371
- "CREATE TABLE statement [UltraLite] [UltraLiteJ]" on page 384

Example

The following statement deletes a fictitious table, EmployeeBenefits, from the database:

```
DROP TABLE EmployeeBenefits;
```

DROP USER statement [UltraLite]

Drops a user. UltraLite only. Not supported for UltraLiteJ.

Syntax

DROP USER *userid*

Parameters

- **userid** The userid of the user you are dropping.

Permissions

DBA authority.

Remarks

None.

Side effects

None.

See also

- "ALTER USER statement [UltraLite]" on page 374
- "CREATE USER statement [UltraLite]" on page 390

Standards and compatibility

- **SQL/2003** Vendor extension.

Example

The following example drops the user SQLTester from a database.

```
DROP USER SQLTester;
```

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. See "Subqueries in expressions" on page 251.

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.

Derived tables
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.

Note
UltraLite does not support KEY JOINS nor NATURAL joins.

See also

- "Joins: Retrieving data from several tables" [*SQL Anywhere Server - SQL Usage*]
- "DELETE statement [UltraLite] [UltraLiteJ]" on page 391
- "SELECT statement [UltraLite] [UltraLiteJ]" on page 402
- "UPDATE statement [UltraLite] [UltraLiteJ]" on page 410

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;
```

Grant Connect to statement [UltraLite]

Use this statement to grant connection permission for a user.

Syntax

GRANT CONNECT TO *userid*.
[**IDENTIFIED BY** *password*]

See also

- "GrantConnectTo method" [*UltraLite - C and C++ Programming*] (C++)
- "GrantConnectTo method" [*UltraLite - .NET Programming*] (.NET)
- "grantConnectTo method" [*UltraLite - M-Business Anywhere Programming*](M-Business Anywhere)

INSERT statement [UltraLite] [UltraLiteJ]

Use this statement to insert 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.

See also

- "SELECT statement [UltraLite] [UltraLiteJ]" on page 402

Example

The following statement adds an Eastern Sales department to the database.

```
INSERT
INTO Departments ( DepartmentID, DepartmentName )
VALUES ( 230, 'Eastern Sales' );
```

LOAD TABLE statement [UltraLite]

Use this statement to import bulk data into a database table from an external file. This statement also provides support for handling the output of the SQL Anywhere dbunload utility (the reload.sql file). LOAD TABLE is not supported on devices (Windows Mobile, iPhone, embedded Linux).

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. The statement to load data from the file *c:\temp\input.dat* into the Employees table is:

  ```
  LOAD TABLE Employees
  FROM '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. If you want 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.

  - **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

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).

LOAD TABLE is only supported for Windows and Linux, not Windows Mobile.

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, '' (single-quote 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.

See also

- "INSERT statement [UltraLite] [UltraLiteJ]" on page 397
- "UltraLite Database Unload utility (ulunload)" on page 214
- "Unload utility (dbunload)" [*SQL Anywhere Server - Database Administration*]

Standards and compatibility

- **SQL/2003** Vendor extension.

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

REVOKE CONNECT FROM statement [UltraLite]

Use this statement to revoke connection permission for a user.

Syntax

REVOKE CONNECT FROM *userid*

See also

- "RevokeConnectFrom method" [*UltraLite - C and C++ Programming*] (C++)
- "RevokeConnectFrom method" [*UltraLite - .NET Programming*] (.NET)
- "revokeConnectFrom method" [*UltraLite - M-Business Anywhere Programming*](M-Business Anywhere)

ROLLBACK statement [UltraLite] [UltraLiteJ]

Use this statement to end a transaction and revert 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.

See also

- "COMMIT statement [UltraLite] [UltraLiteJ]" on page 376

Example

The following statement rolls the database back to the state it was in at the previous commit:

```
ROLLBACK;
```

SELECT statement [UltraLite] [UltraLiteJ]

Use this statement to retrieve 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*n* to return the first *n* rows of the results. Specify START AT*m* to control the location of the starting row when retrieving the TOP*n* 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*. See "FROM clause [UltraLite]" on page 395.

WHERE clause If a WHERE clause is specified, only rows satisfying *search-condition* are selected. See "Search conditions in UltraLite" on page 253.

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.

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. You should specify this clause whenever possible, since 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. See "Using direct page scans" on page 89.

- **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. See "Execution plans in UltraLite" on page 262.

Remarks

Always remember to close the query. Otherwise memory cannot be freed and the number of temporary tables that remain can proliferate unnecessarily.

See also

- "UltraLite performance and optimization" on page 81
- "SELECT statement" [*SQL Anywhere Server - SQL Reference*]
- "Querying data" [*SQL Anywhere Server - SQL Usage*]

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 40th row and ending at the 49th row.

```
SELECT TOP 10 START AT 40 * FROM Employees;
```

SET OPTION statement [UltraLite] [UltraLiteJ]

Use this statement to change the values of database options.

Syntax

SET OPTION *option-name*=*option-value*

option-name: identifier

option-value: string, identifier, or number

Remarks

You can only set database options with this statement and properties cannot be modified after the database has been created. The exception to these rules is**ml_remote_id**.

You cannot specify whether an option is persistent or not. The way an option has been implemented in UltraLite determines whether it is a persistent or temporary option. Persistent options are stored in the sysuldata table. Temporary options are used only until the database stops running.

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.

See also

- “sysuldata system table” on page 224
- “UltraLite database options” on page 162
- “DB_PROPERTY function [System]” on page 299
- “UltraLite ml_remote_id option” on page 166

Example

The following statement sets the global_database_id option to 100:

```
SET OPTION global_database_id=100;
```

START SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]

Use this statement to restart 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. See “UltraLite row states” on page 12.

See also

- "STOP SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 406

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;
```

STOP SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]

Use this statement to stop 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. See "UltraLite row states" on page 12.

See also

- "START SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 405

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;
```

SYNCHRONIZE statement [UltraLite] [UltraLiteJ]

Use this statement to synchronize an UltraLite or UltraLiteJ database via the MobiLink server. The synchronization is configured according to the parameters in the synchronization profile, or the parameters can be specified in the statement itself.

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

See "CREATE SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 379 for a list of the synchronization profile options supported by UltraLite.

See "ALTER SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 369 to understand how sync options are merged with existing options in the synchronization profile.

By allowing sync 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.

See also

- "ALTER SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 369
- "DROP SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ]" on page 393
- "ULSetSynchronizationCallback method" [*UltraLite - C and C++ Programming*]

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'publication=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 'publication=p2;uploadonly=on';
```

TRUNCATE TABLE statement [UltraLite] [UltraLiteJ]

Use this statement to delete 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 Dispose method for all data objects (for example, ULDataReader). See "Dispose method" [*UltraLite - .NET Programming*].

Schema changes Statements are not released if schema changes are initiated at the same time. See "Schema changes with DDL statements" on page 10.

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.

See also

- "DELETE statement [UltraLite] [UltraLiteJ]" on page 391
- "START SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 405
- "STOP SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 406

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. See "ROLLBACK statement [UltraLite] [UltraLiteJ]" on page 402.

UNION statement [UltraLite]

Use this statement to combine 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, note that 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 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.

See also

- "SELECT statement [UltraLite] [UltraLiteJ]" on page 402

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;
```

UPDATE statement [UltraLite] [UltraLiteJ]

Use this statement to modify 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-hand 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. See "Search conditions in UltraLite" on page 253.

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.

See also

- "INSERT statement [UltraLite] [UltraLiteJ]" on page 397
- "DELETE statement [UltraLite] [UltraLiteJ]" on page 391
- "Search conditions in UltraLite" on page 253

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 )
```

UltraLite support for spatial data

This section introduces UltraLite spatial support and explains its purpose, describes the supported data types, and explains how to generate and analyze spatial data.

The spatial data documentation assumes you already have some familiarity with spatial reference systems and with the spatial data you intend to work with. If you do not, links to additional reading material can be found here: "Recommended reading on spatial topics" [*SQL Anywhere Server - Spatial Data Support*].

Introduction to spatial data

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, for instance. 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 UltraLite, points are specified using the ST_Geometry type. See "ST_Geometry type" on page 414.

Compliance and support

This section describes UltraLite's compliance with existing standards and provides a high level view of the supported features.

Compliance with spatial standards

UltraLite spatial support complies with the following standards:

- **International Organization for Standardization (ISO)** UltraLite geometries conform to the ISO standards for defining spatial user-types, routines, schemas, and for processing spatial data. UltraLite conforms to the specific recommendations made by the International Standard ISO/IEC 13249-3:2006. See http://www.iso.org/iso/catalogue_detail.htm?csnumber=38651.

- **Open Geospatial Consortium (OGC) Geometry Model** UltraLite geometries conform to the OGC OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option version 1.2.0 (OGC 06-104r3). See http://www.opengeospatial.org/standards/sfs.

UltraLite uses the standards recommended by the OGC to ensure that spatial information can be shared between different vendors and applications.

To ensure compatibility with UltraLite spatial geometries, it is recommended that you adhere to the standards specified by the OGC.

- **SQL Multimedia (SQL/MM)** UltraLite follows the SQL/MM standard, and uses the prefix **ST_** for all function names.

  SQL/MM is an international standard that defines how to store, retrieve, and process spatial data using SQL. Spatial type hierarchies such as ST_Geometry are one of the functions used to retrieve spatial data. The ST_Geometry hierarchy includes a number of subtypes such as ST_Point, ST_Curve, and ST_Polygon. With the SQL/MM standard, every spatial value included in a query must be defined in the same spatial reference system.

ST_Geometry type

The ST_Geometry type is used to store spatial data in the form of points.

Casts

Geometry objects can be explicitly and implicitly cast to any character or binary type. In the case of a character type, the geometry will be presented in EWKT format. In the case of a binary type, the geometry will be presented in WKB format. Character and binary types can also be explicitly and implicitly cast as a geometry value. For binary values, the value must represent a valid geometry in WKB format. For character values, the value must represent a valid geometry in either WKT or EWKT format. If casting from WKB or WKT to a geometry value, the default value SRID of 0 is assigned to the object.

Column and object definitions

UltraLite provides a fixed set of three different reference systems that can be attributed 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 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.

SRID 4326
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 is supported.

Example

```
create table T1 (
V1 Integer primary key,
V2 ST_Geometry(SRID=0),
V3 ST_Geometry);
```

creates a table with one column associated with the default planar reference system and one with an undefined reference system.

Note
ST_Geometry columns cannot be primary keys.

Functions for spatial data

UltraLite supports the following functions:

ST_AsBinary function

The ST_AsBinary function returns a binary string representing the geometry. The output format is WKB as defined by OGC SFS 1.1. This format does not contain Z and M values.

Syntax

.**ST_AsBinary**(*geometry-expression*)

Returns

- **BINARY** Returns the WKB representation of the *geometry-expression*.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.37

Example

The following returns the result `0x0101000000000000000000f03f0000000000000040`.

```
select ST_AsBinary(ST_Point(1.0, 2.0, 4326))
```

The following returns the result `0x0101000000000000000000f03f0000000000000040`. The server implicitly invokes the ST_AsBinary() function when converting geometries to BINARY.

```
select cast(ST_Point(1.0, 2.0, 4326) as binary(50))
```

ST_AsExtText function

The ST_AsExtText function returns a binary string representing the geometry. The output format is EWKT.

Syntax

.**ST_AsExtText**(*geometry-expression*)

Returns

- **VARCHAR** Returns the EWKT representation of the *geometry-expression*.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.37

Example

The following returns the result `SRID=4326;Point (1 2)`. The SRID is included in the result as a prefix.

```
select ST_AsExtText(ST_Point 1.0, 2.0, 4326))
```

The following returns the result `SRID=4326;Point(1 2)`. The ST_AsExtText() function is implicitly invoked when converting geometries to VARCHAR types.

```
select cast(ST_Point(1.0, 2.0, 4326) as varchar(25))
```

ST_AsText function

The ST_AsText function returns a binary string representing the geometry. The output format is WKT as defined by OGC SFS 1.1.

Syntax

.**ST_AsText**(*geometry-expression*)

Returns

- **VARCHAR** Returns the WKT representation of the *geometry-expression*.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.37

Example

The following returns the result `Point (1 2)`.

```
select ST_AsText(ST_Point(1.0, 2.0, 4326))
```


ST_Distance function

Returns the smallest distance between two specified geometry values. If the points are in SRID 4326, the units are in meters.

Syntax

ST_Distance(*geo1*,*geo2*)

Parameters

Name	Type	Description
geo1	ST_Geometry	The first geometry value to be used to calculate the distance between two geometry values.
geo2	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.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.23

Example

```
Select ST_Distance(ST_Point(-79.38,43.65,4326),ST_Point(-123.1,49.28,4326))
```

Returns `3367142.463`.

ST_Equals function

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.

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.

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.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.24

Example

```
Select ST_Equals(ST_Point(1,1,4326),ST_Point(1,1,4326))
```

Returns `1`.

ST_IntersectsRect function

The ST_IntersectsRect function tests if point is located within the box defined by the two points, min and max.

Syntax

ST_IntersectsRect(*location*,*min*,*max*)

Parameters

Name	Type	Description
location	ST_Geometry	The point to be tested.
min	ST_Geometry	The minimum point value used to define the box.
max	ST_Geometry	The maximum point value used to define the box.

Returns

- **BIT** Returns 1 if *location* intersects with the specified box, otherwise 0.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** Vendor extension

Example

```
Select ST_IntersectsRect(ST_Point(1,1,4326),ST_Point(0,0,4326),
ST_Point(3,3,4326))
```


Returns 1.

ST_Point function

Constructs a point based on x and y coordinates.

Syntax

ST_Point(*x*,*y*,*SRID*)

Parameters

Name	Type	Description
x	DOUBLE	The x coordinate to use to construct the point.
y	DOUBLE	The y coordinate to use to construct the point.
SRID	INTEGER	The SRID value associated with the point.

Returns

- **ST_Point** Returns an ST_Geometry value created from the input string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.8

Example

```
ST_Point(10.0,20.0,2163)
```

Creates a point (10.0,20.0) in the 2163 reference system.

ST_PointFromExtText function

Returns an ST_Geometry value, which is transformed from a VARCHAR value containing the EWKT representation of an ST_Geometry.

Syntax

ST_PointFromText(*ewkt*)

Parameters

Name	Type	Description
ewkt	VARCHAR	The EWKT representation.

Returns

- **ST_Geometry** Returns an ST_Geometry value created from the input string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.8

Example

```
ST_PointFromExtText('SRID=4326;Point(10 20)')
```

Creates a point (10,20) in the 4326 reference system.

ST_PointFromText function

Returns an ST_Geometry value, which is transformed from a VARCHAR value containing the WKT representation of an ST_Geometry.

Syntax

ST_PointFromText(*wkt*)

Parameters

Name	Type	Description
wkt	VARCHAR	The WKT representation.

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*.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.8

Example

```
ST_PointFromText('Point(10 20)',4326)
```

Creates a point (10, 20) in the 4326 reference system.

ST_PointFromWKB function

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.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.9

Example

The following returns point (1.0, 2.0, 4326).

```
ST_PointFromWKB(0x0101000000000000000000f03f0000000000000040,4326)
```

ST_SRID function

Retrieves the spatial reference system (SRID) associated with the geometry value.

Syntax

ST_SRID(geo1, SRID)

Parameters

Name	Type	Description
geo1	ST_Geometry	The point value.
SRID	INTEGER	The SRID value associated with the point.

Returns

- **INT** Returns the SRID of the geometry.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.5

Example

The following returns `4326`.

```
Select ST_SRID(ST_Point(10,20,4326))
```

ST_X function

Returns the x coordinate of the ST_Geometry value.

Syntax

ST_X(geo1)

Parameters

Name	Type	Description
geo1	ST_Geometry	The ST_Geometry value to determine the x value of.

Returns

- **DOUBLE** Returns the x coordinate of the ST_Geometry value.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.3

Example

The following example returns the result `10.0`.

```
SELECT ST_X(ST_Point(10.0,20.0,4326))
```

ST_Y function

Returns the y coordinate of the ST_Geometry value.

Syntax

.**ST_Y**(geo1)

Parameters

Name	Type	Description
geo1	ST_Geometry	The ST_Geometry value to determine the y value of.

Returns

- **DOUBLE** Returns the y coordinate of the ST_Geometry value.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.3

Example

The following example returns the result `20.0`.

```
SELECT ST_Y(ST_Point(10.0,20.0,4326))
```

Troubleshooting UltraLite

Unable to start the UltraLite engine

Symptom

You have 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\uleng12.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\uleng12.exe"
```

or

```
START='"\Program Files\uleng12.exe"'
```

Unable to connect to databases after upgrade

Symptom

You have upgraded UltraLite. You discover that you are able to create an empty UltraLite database using the administration tools. However, when you try to connect to this or any other UltraLite database (including *CustDB.udb*) with Sybase 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 Sybase 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.

See also

- "Upgrading UltraLite" [*SQL Anywhere 12 - Changes and Upgrading*]

Database corruption

Symptom

Your 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 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, 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 database loads a page, UltraLite immediately stops the database and reports a fatal error. This error cannot be corrected. Instead you must:

1. Report the error to iAnywhere. 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 database to a file.
3. Create a new database.
4. Repopulate the data either by synchronizing or by loading the unloaded data.

See also

- "UltraLite checksum_level creation parameter" on page 136

Database size not stabilizing

Symptom

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. This is a concern because file size needs to be managed carefully due to limited resources of the device.

Explanation

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 your UltraLite database that is deployed to the devices.
2. Creating a SQL script of DDL statements that define the schema required by the client devices. See "Deploying UltraLite schema upgrades" on page 51.
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.

See also

- "STOP SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" on page 406
- "TRUNCATE TABLE statement [UltraLite] [UltraLiteJ]" on page 408
- "DELETE statement [UltraLite] [UltraLiteJ]" on page 391
- "ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]" on page 368

Importing ASCII data into a new database

Symptom

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 choose **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).

  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.

See also

- "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "Interactive SQL for UltraLite utility (dbisql)" on page 186

Utilities still running as the previous version

Symptom

You have just installed UltraLite 12. 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. See "Using the utilities" [*SQL Anywhere 12 - Changes and Upgrading*].

Result set changes unpredictably

Symptom

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.

See also

- "SELECT statement [UltraLite] [UltraLiteJ]" on page 402
- "Database corruption" on page 426

UltraLite engine client fails with error -764

Applies to

Windows Mobile

Symptom

You are running the UltraLite engine on Windows Mobile device, and the client returns a -764 error.

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\uleng12.exe")
  ```

  To something similar to:

```
ULConnection conn = new ULConnection(@"dbf=\\\"Program Files \"\
\HelloEngine\\HelloEngine.udb;
START=\\Windows\\uleng12.exe");
```

See also

- “Deploy multiple UltraLite applications with the UltraLite engine” on page 44
- “UltraLite START connection parameter” on page 182

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

