SYBASE iAnywhere

MobiLink™
Server Administration

Contents

About this book

This book describes how to set up and administer MobiLink servers, consolidated databases, and MobiLink applications. It also describes the SQL Anywhere Monitor for MobiLink, a web browser-based administration tool that provides information about the health and availability of MobiLink servers, and the Relay Server, which enables secure communication between mobile devices and MobiLink servers through a web server.

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.

Using MobiLink server technology

This section introduces MobiLink technology and describes how to use it to synchronize data between two or more data sources.

MobiLink consolidated databases

Your consolidated database holds system objects that are required by MobiLink. Usually, it also holds your application data, but you can hold all or part of your application data in other forms as well.

MobiLink supports consolidated databases for Windows and Linux on 32-bit and 64-bit environments. Your consolidated database can be one of the following ODBC-compliant RDBMSs:

- Adaptive Server Enterprise (no 64-bit Linux support provided)
- IBM DB2 LUW
- Microsoft SQL Server
- MySQL
- Oracle
- SQL Anywhere

For version support information, see http://www.sybase.com/detail?id=1002288.

For information about recommended ODBC drivers for MobiLink, see http://www.sybase.com/detail?id=1011880

Your SQL Anywhere installation includes a MobiLink setup script for each type of RDBMS. You need to run the appropriate setup script to use that RDBMS with MobiLink. The setup script adds tables and stored procedures that are required by MobiLink.

For information about setting up each type of RDBMS as a consolidated database, see “Setting up a consolidated database” on page 2.

For information about writing synchronization scripts for particular consolidated databases, see “RDBMS-dependent synchronization scripts” on page 5.

Synchronizing to other data sources

Your MobiLink environment must have a database that has been set up as a consolidated database. However, you can synchronize data sources other than the consolidated database. The other data sources can be almost anything: a text file, web service, non-relational database, spreadsheet, and so on. You can:

- Synchronize to only a consolidated database.
- Synchronize to only another data source.
- Create a hybrid application in which you synchronize to both a consolidated database and some other data source.

See "Direct row handling" on page 671.

Restrictions on modifying your consolidated database

Some users have limited ability to change the schema of their consolidated database. For these situations, MobiLink provides solutions, where possible, to keep changes to the consolidated database to a minimum. For example, MobiLink offers a variety of solutions for maintaining unique primary keys across the synchronization system, some of which have minimal impact on the consolidated database schema.

In addition, you can avoid almost all impact on your consolidated database by putting your MobiLink system objects in a separate database. See "MobiLink system database" on page 4.

How remote tables relate to consolidated tables

Your synchronization design specifies mappings between tables and columns in the remote databases with tables and columns in the consolidated database. Typically, tables and columns in remote databases either exactly match the tables and columns in the consolidated database or are subsets of them.

Arbitrary relationships permitted

Tables in a remote database need not be identical to those in the consolidated database. Synchronized data in one remote application table can be distributed between columns in different tables. You specify these relationships using synchronization scripts.

Direct relationships are simple

The simplest and most common design uses a table structure in the remote database that is a subset of that in the consolidated database. Using this design, every table in the remote database exists in the consolidated database. Corresponding tables have the same structure and foreign key relationships as those in the consolidated database.

The consolidated database frequently contains columns and tables that are not synchronized. Some of these columns or tables may be used to assist synchronization. For example, a timestamp column can identify new or updated rows in the consolidated database; or a shadow table can be used to track deletes. Non-synchronized columns or tables in the consolidated database can also hold information that is not required at remote sites.

Remote databases also frequently contain tables or columns that aren't synchronized.

See also

- "MobiLink data mappings between remote and consolidated databases" on page 737

Setting up a consolidated database

Setup scripts

To set up a database so that it can be used as a MobiLink consolidated database, you must run a setup script. Your SQL Anywhere installation includes a script for each of the supported RDBMSs. These

scripts are all located in *install-dir\MobiLink\setup*. You can also use the following methods to update the MobiLink system setup:

- In the MobiLink 12 plug-in for Sybase Central, choose **Folders** from the **View** menu. Open your MobiLink project and expand **Consolidated Databases**. Right-click the database name and choose **Check MobiLink System Setup**. If your database requires setup, you are prompted to continue.

- When you use the **Create Synchronization Model Wizard** or **Deploy Synchronization Model Wizard**, the system setup is checked when you connect to your server database. If your database requires setup, you are prompted to continue. See "Introduction to synchronization models" [*MobiLink - Getting Started*].

The MobiLink setup script adds MobiLink system tables, stored procedures, triggers, and views to your database. These tables and procedures are required for MobiLink synchronization.

For information about the stored procedures that are installed, see "MobiLink server system procedures" on page 691.

Note
The database user who runs the setup scripts is expected to be the same one used to update the MobiLink system tables during synchronization. This users must be used to start the MobiLink server and to configure MobiLink. See "Required permissions" on page 22.

For instructions on how to run the setup scripts, see the section for your RDBMS:

- "Adaptive Server Enterprise consolidated database" on page 9
- "IBM DB2 LUW consolidated database" on page 10
- "Microsoft SQL Server consolidated database" on page 13
- "MySQL consolidated database" on page 14
- "Oracle consolidated database" on page 17
- "SQL Anywhere consolidated database" on page 20

Note
If the consolidated database you are setting up is to be used as a QAnywhere Server Store, the database should be configured to be case insensitive for comparisons and string operations.

ODBC connection

The MobiLink server needs an ODBC connection to your consolidated database. You must configure the appropriate ODBC driver for your RDBMS and create an ODBC data source for the database on the computer where your MobiLink server is running.

For more information about MobiLink ODBC drivers, see "iAnywhere Solutions ODBC drivers for MobiLink" on page 778.

For updated information and complete functional specifications of the ODBC drivers you can use with MobiLink, see http://www.sybase.com/detail?id=1011880.

MobiLink system database

In some rare cases, you may want to split your consolidated database into two: one database for data and one for the MobiLink system information. When you do this you do not have to add MobiLink system objects to your consolidated database. All MobiLink system objects can be stored in a separate database called the MobiLink system database.

Note
This setup requires a distributed transaction coordinator. Currently the only one supported is by MobiLink is Microsoft Distributed Transaction Coordinator (MS DTC), which only runs on Windows.

Your MobiLink system database can be any database that is supported as a consolidated database. It does not have to be the same RDBMS as your consolidated database.

It is easy to set up a MobiLink system database. Simply apply MobiLink setup scripts to a database other than your consolidated database. When you start the MobiLink server, connect to both databases using -c for the consolidated and -cs for the system database.

Notes

- If you are using a separate system database, you can only run the MobiLink server on Windows using Microsoft Distributed Transaction Coordinator.
- You cannot use a MobiLink system database with the MobiLink **Deploy Synchronization Model Wizard**.
- There is a performance penalty for storing MobiLink system objects in a separate database.

MobiLink server system tables

MobiLink system tables store information about MobiLink users, subscriptions, tables, scripts, script versions, and other information. They are required for MobiLink synchronization. Although you can modify the MobiLink system tables, you usually do not need to.

MobiLink system tables are created when you run the MobiLink setup script for your consolidated database. They must be stored on your consolidated database, and are always prefixed with **ml_**. The database user who runs the setup script is the owner of the MobiLink system tables that are created by the script.

See “Setting up a consolidated database” on page 2.

Notes

- This section provides data types for the MobiLink system tables in SQL Anywhere consolidated databases. In some RDBMSs, the data types are slightly different.

RDBMS-dependent synchronization scripts

MobiLink uses synchronization scripts to define the rules you use to synchronize data. Synchronization scripts define:

- How data uploaded from the remote database is to be applied to the consolidated database.
- What data should be downloaded from the consolidated database to the remote database.

See "Writing synchronization scripts" on page 263.

For a complete list of events you can write scripts for, see "Synchronization events" on page 297.

For specific information about each type of consolidated database, see:

- "SQL Anywhere consolidated database" on page 20
- "Adaptive Server Enterprise consolidated database" on page 9
- "IBM DB2 LUW consolidated database" on page 10
- "Microsoft SQL Server consolidated database" on page 13
- "MySQL consolidated database" on page 14
- "Oracle consolidated database" on page 17

.NET and Java synchronization scripts

You can write your synchronization logic in the version of the SQL language used by your database. You can also write more portable and powerful scripts using Java or .NET. Both Java and .NET offer flexibility beyond what each RDBMS provides via SQL, while also providing full SQL compatibility. When you use Java or .NET synchronization logic, you can hold session-wide variables, create user-defined procedures, integrate authentication to external servers, and so on.

For information about Java synchronization logic, see "Writing Java synchronization logic" on page 525.

For information about .NET synchronization logic, see "Writing synchronization scripts in .NET" on page 588.

Invoking procedures from scripts

Some databases, such as Microsoft SQL Server, require that procedure calls with parameters be written using the ODBC syntax.

```
{ CALL procedure_name( {ml param1}, {ml param2}, ... ) }
```

You can return values by defining the parameters as OUT or INOUT in the procedure definition.

CHAR columns

In many other RDBMSs, CHAR data types are fixed length and blank-padded to the full length of the string. In SQL Anywhere or UltraLite remote MobiLink databases, CHAR is the same as VARCHAR: values are not blank-padded to a fixed width. If you are not using SQL Anywhere as your consolidated database, It is strongly recommended that you use VARCHAR in the consolidated database rather than CHAR. If you must use CHAR, the mlsrv12 -b command line option can be used to remove trailing

blanks from strings during synchronization. This option is important for string comparisons used to detect conflicts.

See "-b mlsrv12 option" on page 38.

Data conversion

For information about the conversion of data that must take place when a MobiLink server communicates with a consolidated database that isn't SQL Anywhere, see "MobiLink data mappings between remote and consolidated databases" on page 737.

Synchronizing spatial data

The MobiLink server supports synchronization of tables containing columns with spatial data types. The following types of consolidated databases are supported for spatial data synchronization:

- SQL Anywhere
- Oracle
- Microsoft SQL Server
- IBM DB2 LUW
- MySQL

The MobiLink server uses a Well Known Binary (WKB) representation of spatial data along with its Spatial Reference Identifier (SRID) to upload data from remote databases to a consolidated database. It also uses WKB and SRID for downloading spatial data from the consolidated database to the remote databases. Therefore, the upload and download table scripts must be able to handle spatial data in the WKB format.

Dimensional restrictions

The MobiLink server is able to synchronize two-dimensional, three-dimensional and four-dimensional spatial data between the consolidated and remote databases if the consolidated database is running on a SQL Anywhere server. However, the MobiLink server is only able to synchronize two-dimensional data between the consolidated and remote databases if the consolidated database is one of the other supported RDBMS types.

If an upload stream contains three-dimensional and/or four-dimensional dimensional spatial data and the consolidated database is running on a non-SQL Anywhere database server, the MobiLink server generates a warning message and then drops the third and/or fourth dimension value before sending the spatial data to the consolidated database.

SRID requirements

The MobiLink server does not automatically find a mapping between the SRIDs used in the consolidated and remote databases. You must make sure that either the SRIDs used in the remote databases match those defined in the consolidated databases, or user defined upload and download table scripts must be

able to convert the SRIDs back and forth between the SRIDs defined in the consolidated and remote databases.

Named parameters

Upload table scripts can be written using named parameters or question marks (?). The use of question marks for SQL scripts is deprecated and it is recommended that you use named parameters. If question marks are used, each spatial column must contain two questions marks. The first one is for the WKB value and the second is for the SRID value.

If named parameters are used, the named parameters must have the following convention:

```
{ml r.column_name:data}

{ml r.column_name:srid}
```

The first named parameter represents the WKB value and the second named parameter is for the SRID.

See also

- "Introduction to spatial data" [*SQL Anywhere Server - Spatial Data Support*]
- "Named script parameters" on page 268
- "User-defined named parameters" on page 280

Upload and download scripts

When a table contains spatial columns, the upload and download scripts may vary greatly depending on the type of consolidated database being used. The following examples show some sample upload and download scripts for spatial data for the consolidated databases currently supported by the MobiLink server.

For the examples below, the synchronization table is assumed to be as follows:

```
create table test (c1 int not null primary key, c2 st_geometry )
```

SQL Anywhere sample scripts

The following is a sample upload_insert script for SQL Anywhere:

```
INSERT INTO test VALUES( {ml r.c1}, ST_Geometry::ST_GeomFromBinary({ml
r.c2:data},{ml r.c2:srid}) )
```

The following is a sample download_cursor script for SQL Anywhere:

```
SELECT c1, c2.ST_AsBinary(), c2.ST_SRID() FROM test
```

Microsoft SQL Server sample scripts

The following is a sample upload_insert script for Microsoft SQL Server:

```
BEGIN
      DECLARE @c1 INTEGER
      DECLARE @v1 VARBINARY(max)
      DECLARE @s1 INTEGER
      DECLARE @g1 geometry
      SELECT @c1 = {ml r.c1}
      SELECT @v1 = {ml r.c2:data}
```

```
      SELECT @s1 = {ml r.c2:srid}
      IF @v1 IS NULL
            SELECT @g1 = NULL
      ELSE
            SELECT @g1 = Geometry::STGeomFromWKB(@v1,@s1)
      INSERT INTO test VALUES( @c1, @g1 )
END
```

The following is a sample download_cursor script for Microsoft SQL Server:

```
SELECT c1, c2.STAsBinary(), c2.STSrid FROM test
```

Oracle sample scripts

The following is a sample upload_insert script for Oracle:

```
DECLARE
     v_c1 INTEGER;
     v_v1 sdo_geometry;
     v_s1 INTEGER;
     v_u1 blob;
BEGIN
     v_c1 := {ml r.c1};
     v_u1 := {ml r.c2:data};
     v_s1 := {ml r.c2:srid};
     IF v_u1 IS NULL THEN
           v_v1 := NULL;
     ELSE
           v_v1 := sdo_geometry( v_u1, v_s1 );
     END IF;
     INSERT INTO test VALUES( v_c1, v_v1 );
END;
```

The following is a sample download_cursor script for Oracle:

```
SELECT c1, g.c2.Get_WKB(), g.c2.sdo_srid FROM test g
```

IBM DB2 sample scripts

The following is a sample upload_insert script for IBM DB2:

```
BEGIN ATOMIC
      DECLARE v_c1 INTEGER;
      DECLARE v_v1 BLOB(10m);
      DECLARE v_s1 INTEGER;
      SET v_c1 = {ml r.c1}; SET v_v1 = {ml r.c2:data}; SET v_s1 = {ml
r.c2:srid};
      INSERT INTO test VALUES( v_c1, ST_Geometry( v_v1, v_s1 ) );
END
```

The following is a sample download_cursor script for IBM DB2:

```
SELECT c1, ST_AsBinary( c2 ), ST_SRID( c2) FROM test
```

MySQL sample scripts

The following is a sample upload_insert script for MySQL:

```
INSERT INTO test VALUES( {ml r.c1}, GeometryFromWKB({ml r.c2:data},{ml
r.c2:srid}) )
```

The following is a sample download_cursor script for MySQL:


```
SELECT c1, AsBinary( c2 ), SRID( c2 ) FROM test
```

For specific information about each type of consolidated database, see:

- "SQL Anywhere consolidated database" on page 20
- "Microsoft SQL Server consolidated database" on page 13
- "MySQL consolidated database" on page 14
- "Oracle consolidated database" on page 17

Adaptive Server Enterprise consolidated database

Permissions

Before running the setup script, you should be aware of the following requirements:

- The database user who runs the setup script is expected to be the same one used to update the MobiLink system tables during synchronization. This user must be used to start the MobiLink server and to configure MobiLink applications. See "Required permissions" on page 22.
- The RDBMS user that the MobiLink server uses to connect to the consolidated database must be able to use the MobiLink system tables, procedures, and so on, without any qualifiers (for example, SELECT * from ml_user). See "MobiLink server system tables" on page 4.
- The MobiLink server login ID must have a SELECT permission permission on MASTER..SYSTRANSACTIONS.
- The MobiLink server login ID must have the dtm_tm_role role, if the -cs option for mlsrv12 is used.

Setting up Adaptive Server Enterprise as a consolidated database

To set up Adaptive Server Enterprise to work as a MobiLink consolidated database, you must run a setup procedure that adds MobiLink system tables, stored procedures, triggers, and views that are required for MobiLink synchronization. There are multiple ways you can do this:

- Run the *syncase.sql* setup script, located in *install-dir\MobiLink\setup*.
- In the MobiLink 12 plug-in for Sybase Central, choose **Folders** from the **View** menu. Open your MobiLink project and expand **Consolidated Databases**. Right-click the database name and choose **Check MobiLink System Setup**. If your database requires setup, you are prompted to continue.
- When you use the **Create Synchronization Model Wizard** or **Deploy Synchronization Model Wizard**, the system setup is checked when you connect to your server database. If your database requires setup, you are prompted to continue.

ODBC driver

You must set up an ODBC DSN for your Adaptive Server Enterprise consolidated database using the ODBC driver that is provided with your Adaptive Server Enterprise database. See:

- http://www.sybase.com/detail?id=1011880
- Your Adaptive Server Enterprise documentation

Adaptive Server Enterprise considerations

- **Data type mapping** The data types of columns must map correctly between your consolidated and remote database. See "Adaptive Server Enterprise data mapping" on page 737.

- **CHAR columns** In Adaptive Server Enterprise, CHAR data types are fixed length and blank-padded to the full length of the string. In MobiLink remote databases (SQL Anywhere or UltraLite), CHAR is the same as VARCHAR: values are not blank-padded to a fixed width. It is strongly recommended that you use VARCHAR in the consolidated database rather than CHAR. If you must use CHAR, the mlsrv12 -b command line option can be used to remove trailing blanks from strings during synchronization. This option is important for string comparisons used to detect conflicts.

  For more information, see "-b mlsrv12 option" on page 38.

- **BLOB sizes** To download BLOB data with a data size greater than 32 KB (the default), do the following:

  - On Windows, set **Text Size** on the **Advanced** page of the **Adaptive Server Enterprise ODBC Driver Configuration** window to be greater than the largest expected BLOB.
  - On Linux, set the TextSize entry in the *obdc.ini* file to be greater than the largest expected BLOB.

- **Restrictions on VARBIT** MobiLink does not support synchronizing 0 length (empty) VARBIT or LONG VARBIT values to an Adaptive Server Enterprise consolidated database. Adaptive Server Enterprise does not support a VARBIT type so these types would normally be synchronized to a VARCHAR or TEXT column in the Adaptive Server Enterprise database. On Adaptive Server Enterprise, empty string values are converted into a single space. A space is not allowed in a VARBIT column on SQL Anywhere, so an attempt to download these values causes an error on the remote database.

Isolation level

See "MobiLink isolation levels" on page 119.

IBM DB2 LUW consolidated database

MobiLink supports IBM DB2 LUW for Linux, Unix, and Windows.

Permissions

Before running the setup script, you should be aware of the following requirements:

- The database user who runs the setup script is expected to be the same one used to update the MobiLink system tables during synchronization. This user must be used to start the MobiLink server and to configure MobiLink applications. See "Required permissions" on page 22.
- The RDBMS user that the MobiLink server uses to connect to the consolidated database must be able to use the MobiLink system tables, procedures, and so on, without any qualifiers (for example, SELECT * from ml_user). See "MobiLink server system tables" on page 4.

Setting up IBM DB2 LUW as a consolidated database

To set up IBM DB2 to work as a MobiLink consolidated database, you must run a setup procedure that adds MobiLink system tables, stored procedures, triggers, and views that are required for MobiLink synchronization. There are multiple ways you can do this:

- Run the *syncdb2.sql* setup script, located in *install-dir\MobiLink\setup*. Before running the file, you must copy it to another location and modify it. Instructions follow.
- In the MobiLink 12 plug-in for Sybase Central, choose **Folders** from the **View** menu. Open your MobiLink project and expand **Consolidated Databases**. Right-click the database name and choose **Check MobiLink System Setup**. If your database requires setup, you are prompted to continue. Note that you are required to perform step 1 of the following procedure.
- When you use the **Create Synchronization Model Wizard** or **Deploy Synchronization Model Wizard**, the system setup is checked when you connect to your consolidated database. If your database requires setup, you are prompted to continue. Note that you are required to perform step 1 of the following procedure.

To run the IBM DB2 setup script

1. To install MobiLink system tables using the setup script, the targeted IBM DB2 LUW tablespace must use a minimum of 8 KB pages. If a tablespace does not use 8 KB pages, complete the following steps:
   - Verify that at least one of your buffer pools has 8 KB pages. If not, create a buffer pool with 8 KB pages.
   - Create a new tablespace and temporary tablespace that use the buffer pool with 8 KB pages.

     For more information, consult your IBM DB2 LUW documentation.
2. Customize *syncdb2.sql* with your connection information:
   a. Copy *syncdb2.sql* to a new location where it can be modified and stored.
   b. The *syncdb2.sql* script contains a default connection statement, `connect to DB2Database`. Alter this line to connect to your IBM DB2 database. Use the following syntax:

      ```
      connect to DB2Database user userid using password ~
      ```

      where *DB2Database*, *userid*, and *password* are names you provide. (The *syncdb2.sql* script uses the tilde character (~) as a command delimiter.)
3. Run *syncdb2.sql*:

   ```
   db2 -c -ec -td~ +s -v -f syncdb2.sql
   ```

ODBC driver

You must set up an ODBC DSN for your IBM DB2 consolidated database using the ODBC driver that is provided with your IBM DB2 database. See:

- http://www.sybase.com/detail?id=1011880
- IBM DB2 LUW documentation

IBM DB2 LUW considerations

- **Data type mapping** The data types of columns must map correctly between your consolidated and remote database. For details, see "IBM DB2 LUW data mapping" on page 746.

- **CHAR columns** In IBM DB2 LUW, CHAR data types are fixed length and blank-padded to the full length of the string. In MobiLink remote databases (SQL Anywhere or UltraLite) CHAR is the same as VARCHAR: values are not blank-padded to a fixed width. It is strongly recommended that you use VARCHAR in the consolidated database rather than CHAR. If you must use CHAR, the mlsrv12 -b command line option can be used to remove trailing blanks from strings during synchronization. This option is important for string comparisons used to detect conflicts.

  See "-b mlsrv12 option" on page 38.

- **Tablespace capacity** A tablespace and temporary tablespace of any IBM DB2 LUW database that you want to use as a consolidated database must use at least 8 KB pages.

  In addition, there are columns that require a LONG tablespace. If there is no default LONG tablespace, the creation statements for the tables containing these columns must be qualified appropriately, as in the following example:

  ```
  CREATE TABLE ... ( ... )
  IN tablespace
  LONG IN long-tablespace
  ```

  For an example using the sample application, see "Exploring the CustDB sample for MobiLink" [*MobiLink - Getting Started*].

- **Session-wide variables** IBM DB2 LUW before version 8 does not support session-wide variables. A convenient solution is to use a base table with columns for the MobiLink user name and other session data. The base table has rows representing concurrent synchronizations.

- **User-defined procedures** IBM DB2 LUW before version 8.2 requires that you compile SQL procedures into an executable library (such as a DLL). The resulting DLL/shared library must be copied to a special directory on the server. Note that you can write procedures using several different languages, including C/C++ and Java, among others.

  For more information about Java and .NET synchronization scripts, see:

  - "Writing synchronization scripts in Java" on page 523
  - "Writing synchronization scripts in .NET" on page 588

- **Double up the quotation marks in system procedure calls** When you use a MobiLink system procedure to add scripts to your IBM DB2 consolidated database, you need to double up the quotation

marks. For example, if the script you are adding with ml_add_table_script includes the line `SET "DELETED"=''Y''` for any other consolidated database, for IBM DB2 you would have to write this as `SET "DELETED" = ''''Y''''`.

Isolation level

See "MobiLink isolation levels" on page 119.

Microsoft SQL Server consolidated database

Permissions

Before running the setup script, you should be aware of the following requirements:

- The database user that runs the setup script must be able to create tables, triggers, and stored procedures, so must have the db_owner role.
- The database user who runs the setup script is expected to be the same one used to update the MobiLink system tables during synchronization. This user must be used to start the MobiLink server and to configure MobiLink applications. See "Required permissions" on page 22.
- The RDBMS user that the MobiLink server uses to connect to the consolidated database must be able to use the MobiLink system tables, procedures, and so on, without any qualifiers (for example, SELECT * from ml_user). See "MobiLink server system tables" on page 4.
- The MobiLink server login ID must have VIEW SERVER STATE permission. This permission can be granted using the following SQL statement within the master database of a Microsoft SQL server:

  ```
  grant view server state to user_name
  ```

- The MobiLink server login ID must have SELECT permission on sys.databases.

Setting up Microsoft SQL Server as a consolidated database

To set up Microsoft SQL Server to work as a MobiLink consolidated database, you must run a setup procedure that adds MobiLink system tables, stored procedures, triggers, and views that are required for MobiLink synchronization. There are multiple ways you can do this:

- Run the *syncmss.sql* setup script, located in *install-dir\MobiLink\setup*.
- In the MobiLink 12 plug-in for Sybase Central, choose **Folders** from the **View** menu. Open your MobiLink project and expand **Consolidated Databases**. Right-click the database name and choose **Check MobiLink System Setup**. If your database requires setup, you are prompted to continue. If you want to use an existing MobiLink system setup, then your default_schema should be the schema of the MobiLink system setup.
- When you use the **Create Synchronization Model Wizard** or **Deploy Synchronization Model Wizard**, the system setup is checked when you connect to your server database. If your database requires setup, you are prompted to continue.

ODBC driver

You must set up an ODBC DSN for your SQL Server consolidated database using the ODBC driver that is provided with your SQL Server database. See:

- http://www.sybase.com/detail?id=1011880
- Microsoft SQL Server documentation

SQL Server considerations

- **Data type mapping** The data types of columns must map correctly between your consolidated and remote database. For details, see "Microsoft SQL Server data mapping" on page 754.

- **CHAR columns** In Microsoft SQL Server, CHAR data types are fixed length and blank-padded to the full length of the string. In MobiLink remote databases (SQL Anywhere or UltraLite) CHAR is the same as VARCHAR: values are not blank-padded to a fixed width. It is recommended that you use VARCHAR in the consolidated database rather than CHAR. If you must use CHAR, the mlsrv12 -b command line option can be used to remove trailing blanks from strings during synchronization. This option is important for string comparisons used to detect conflicts.

  See "-b mlsrv12 option" on page 38.

- **SET NOCOUNT ON** For Microsoft SQL Server, you should specify SET NOCOUNT ON as the first statement in all stored procedures or SQL batches executed via ODBC.

- **Procedure calls** Microsoft SQL Server requires that procedure calls with parameters be written using the ODBC syntax:

  ```
  { CALL procedure_name( {ml param1}, {ml param2}, ... ) }
  ```

- **Sample database uses computer columns** The SQL Server AdventureWorks sample database contains computed columns. You cannot upload to a computed column. You can set the column to be download-only, or you can exclude the column from synchronization.

- **Implementing conflict detection in an upload_update script** The behavior of the SQL Server NOCOUNT option means that the MobiLink server cannot accurately assess how many rows were changed by an upload data script. For SQL Server, you must perform conflict detection directly in the upload_update script by passing in both the old and the new row values. See "upload_update table event" on page 519.

Isolation level

See "MobiLink isolation levels" on page 119.

MySQL consolidated database

The MobiLink server supports MySQL Community and Enterprise servers 5.1.3 or later. QAnywhere does not support MySQL.

Permissions

Before running the setup script, you should be aware of the following requirements:

- The database user who runs the setup script is expected to be the same one used to update the MobiLink system tables during synchronization. This user must be used to start the MobiLink server and to configure MobiLink applications. See "Required permissions" on page 22.

- The RDBMS user that the MobiLink server uses to connect to the consolidated database must be able to use the MobiLink system tables, procedures, and so on, without any qualifiers (for example, SELECT * from ml_user). See "MobiLink server system tables" on page 4.

Setting up MySQL as a consolidated database

To set up MySQL to work as a MobiLink consolidated database, you must run a setup procedure that adds MobiLink system tables, stored procedures, triggers, and views that are required for MobiLink synchronization. There are two ways you can do this:

- Using the MySQL command line tool or the MySQL Query Browser, run the *syncmys.sql* setup script, located in *install-dir\MobiLink\setup*. Make sure that your MySQL user ID has privileges to create triggers.

- In the MobiLink 12 plug-in for Sybase Central, choose **Folders** from the **View** menu. Open your MobiLink project and expand **Consolidated Databases**. Right-click the database name and choose **Check MobiLink System Setup**. If your database requires setup, you are prompted to continue. Note that if you want to use an existing MobiLink system setup, then your default_schema should be the schema of the MobiLink system setup.

ODBC driver

You must set up an ODBC DSN for your MySQL consolidated database using the ODBC driver that is provided on the MySQL web site. The MobiLink server supports MySQL ODBC driver 5.1.3 or later. See:

- http://www.sybase.com/detail?id=1011880

To specify your ODBC configuration file in Unix, do one of the following,

- Place the *ODBC.INI* file into the home directory of the current user.
- Create an **ODBCINI** environment variable and set it to the directory location of the *ODBC.INI* file.

MySQL considerations

- **Data type mappings** The types of columns must map correctly between your consolidated and remote databases. For details, see "MySQL data mapping" on page 761.

- **Storage Engine** The MobiLink server requires the default storage engine to be ACID compliant. If the default storage engine is not ACID compliant, make sure that all MobiLink server tables are created using an ACID compliant storage engine, such as InnoDB and Falcon.

- **Stored Procedures** You cannot use INOUT or OUT parameters in stored procedure calls. Procedures that require these parameters must be implemented as functions that return an OUT value.

Server events that require INOUT parameters, such as authenticate_user and modify_user, must be implemented as functions and run using a SELECT statement instead of a CALL statement.

- **Named parameters** User-defined named parameters are not supported.

- **Cursor Scripts** The events upload_fetch, download_cursor, and download_delete_cursor must be called using a SELECT statement, which the MobiLink server runs using a read-committed isolation level. A bug in the MySQL ODBC driver causes the server to read uncommitted operations, such as INSERT, UPDATE, and DELETE statements, which results in inconsistent data between the consolidated database and the remote database.

  To work around this problem, affix a LOCK IN SHARE MODE clause to your SELECT statements. For example,

  ```
  SELECT column1 FROM table1 WHERE id > 0 LOCK IN SHARE MODE
  ```

  This clause protects the SELECT statement from uncommitted operations.

- **Bulk upload** The MySQL ODBC driver does not currently support bulk upload.

- **MLSD** The MySQL ODBC driver does not currently support MSDTC, so a separate MLSD is not supported.

- **SQLLEN Datatypes on the 64-bit MobiLink server for Unix** The MySQL ODBC driver defines SQLLEN as a 32-bit integer, causing a discrepancy with the 64-bit MobiLink server, which defines SQLLEN as a 64-bit integer by default. If you are running MobiLink on a 64-bit Unix environment, you must add the following to your ODBC configuration file,

  ```
  length32=1
  ```

  This entry forces the server to read SQLLEN as a 32-bit integer. Your configuration should look similar to the following example,

  ```
  [a_mysql_dsn]
  Driver=full_path/libmyodbc5.so
  server=host_name
  uid=user_name
  pwd=user_password
  database=database_name
  length32=1
  ```

- **MySQL Server Configuration** The MobiLink synchronization scripts are stored in the ml_script table as TEXT and are retrieved when needed. You may need to set max_allowed_packet equal to 16m or greater in the *my.ini* file.

- **Conflict detection** The scripts generated for conflict resolution with a MySQL consolidated database have multiple statements. If you are using conflict detection, you must enable the **Allow Multiple Statements** checkbox on the **Flags 3** page of the **MySQL Connector/ODBC Data Source Configuration** window when you configure a DSN for the MobiLink server to make connections to your MySQL database.

- **Multiple statements** If any of your synchronization scripts contain batched SQL commands separated by semicolons, you may need to select the **Allow Multiple Statements** checkbox on the

Flags 3 page of the **MySQL Connector/ODBC Data Source Configuration** window when you configure a DSN for the MobiLink server to make connections to your MySQL database.

Isolation level

See "MobiLink isolation levels" on page 119.

Oracle consolidated database

Permissions

Before running the setup script, you should be aware of the following requirements:

- The database user who runs the setup script is expected to be the same one used to update the MobiLink system tables during synchronization. This user must be used to start the MobiLink server and to configure MobiLink applications. See "Required permissions" on page 22.
- The RDBMS user that the MobiLink server uses to connect to the consolidated database must be able to use the MobiLink system tables, procedures, and so on, without any qualifiers (for example, SELECT * from ml_user). See "MobiLink server system tables" on page 4.
- The MobiLink server login ID must have a SELECT permission on gv_$transaction and EXECUTE permission on sys.dbms_utility. These permissions can be granted using the Oracle grant permission syntax to the MobiLink server login ID. You must connect as SYS to grant this access. The Oracle syntax for granting this access is:

```
grant select on SYS.GV$TRANSACTION to user-name;
grant execute on SYS.DBMS_UTILITY to user-name;
```

Setting up Oracle as a consolidated database

To set up Oracle to work as a MobiLink consolidated database, you must run a setup procedure that adds MobiLink system tables, stored procedures, triggers, and views that are required for MobiLink synchronization. There are multiple ways you can do this:

- Run the *syncora.sql* setup script, located in *install-dir\MobiLink\setup*.
- In the MobiLink 12 plug-in for Sybase Central, choose **Folders** from the **View** menu. Open your MobiLink project and expand **Consolidated Databases**. Right-click the database name and choose **Check MobiLink System Setup**. If your database requires setup, you are prompted to continue. Note that if you have set up aliases for an existing MobiLink system setup, you should connect as the user whose schema has the MobiLink system setup.
- When you use the **Create Synchronization Model Wizard** or **Deploy Synchronization Model Wizard**, the system setup is checked when you connect to your server database. If your database requires setup, you are prompted to continue.

ODBC driver

You must set up an ODBC DSN for your Oracle consolidated database. See:

- "iAnywhere Solutions 12 - Oracle ODBC driver" on page 779
- http://www.sybase.com/detail?id=1011880

Oracle considerations

- **Data type mapping** The data types of columns must map correctly between your consolidated and remote database. For details, see "Oracle data mapping" on page 767.

- **CHAR columns** In Oracle, CHAR data types are fixed length and blank-padded to the full length of the string. In MobiLink remote databases (SQL Anywhere or UltraLite), CHAR is the same as VARCHAR: values are not blank-padded to a fixed width. It is strongly recommended that you use VARCHAR in the consolidated database rather than CHAR. If you must use CHAR, the mlsrv12 -b command line option can be used to remove trailing blanks from strings during synchronization. This option is important for string comparisons used to detect conflicts.

  See "-b mlsrv12 option" on page 38.

- **Timestamps** The MobiLink server uses gv$transaction to generate a timestamp for the remote to be used in the next synchronization, so the MobiLink server login ID must have a SELECT permission on gv$transaction.

- **Stored procedures** If you are using stored procedures to return result sets or accept VARRAY parameters, you must select the **Procedure returns results or uses VARRAY parameters** option for the iAnywhere Solutions 12 - Oracle ODBC driver. Also, Sybase Central requires procedures to return results in order to use central administration of remote databases, so this option needs to be selected when using central administration.

  See "iAnywhere Solutions 12 - Oracle ODBC driver" on page 779.

- **Session-wide variables** You can store session-wide information in variables within Oracle packages. Oracle packages allow variables to be created, modified and destroyed; these variables may last as long as the Oracle package is current.

- **Autoincrement methods** To maintain primary key uniqueness, you can use an Oracle sequence to generate a list of keys similar to that of a SQL Anywhere autoincrement field. The CustDB sample database provides coding examples, which can be found in *Samples\MobiLink\CustDB\custora.sql*. Unlike autoincrement, however, you must explicitly reference the sequence. SQL Anywhere autoincrement inserts a column value automatically if the column is not referenced in an INSERT statement.

- **Oracle does not support empty strings** In Oracle, an empty string is treated as null. In SQL Anywhere and UltraLite, empty strings have a different meaning from null. Therefore, you should avoid using empty strings in your client databases when you have an Oracle consolidated database.

Isolation level

See "MobiLink isolation levels" on page 119.

Using Oracle VARRAY

The iAnywhere Solutions 12 - Oracle ODBC driver supports the use of Oracle VARRAY in stored procedures. Using VARRAY in upload scripts (upload_insert, upload_update, and upload_delete) that are written in stored procedures may improve performance of the MobiLink server, compared with upload scripts written in stored procedures that do not use VARRAY. Simple SQL statements such as INSERT, UPDATE and DELETE without stored procedures usually offer the best performance, however using stored procedures, including the VARRAY technique, provides an opportunity to apply business logic that the simple statements do not.

VARRAY example

The following is a simple example that uses VARRAY:

1. Create a table called my_table that contains 3 columns.

```
create table my_table ( pk integer primary key, c1 number(20), c2
varchar2(4000) )
```

2. Create user-defined collection types using VARRAYs.

```
create type my_integer is varray(100) of integer;
create type my_number is varray(100) of number(20);
create type my_varchar is varray(100) of varchar2(8000);
```

   my_varchar is defined as a VARRAY that contains 100 elements and each element is a data type of varchar2 and width of 8000. The width is required to be twice as big as that specified for my_table.

3. Create stored procedures for insert.

```
create or replace procedure my_insert_proc( pk_v my_integer, c1_v
my_number, c2_v my_varchar )
is
c2_value my_varchar;
begin
      c2_value := c2_v;  -- Work around an Oracle bug
      FORALL i in 1 .. pk_v.COUNT
            insert into my_table ( pk, c1, c2 ) values( pk_v(i), c1_v(i),
c2_value(i) );
end;
```

VARRAY restrictions

The following restrictions apply when using VARRAY in stored procedures:

- The ODBC data source must have the **Enable Microsoft distributed transactions** checkbox unchecked.
- BLOB and CLOB VARRAYs are not supported.
- If VARRAY is a data type of CHAR, VARCHAR, NCHAR or NVARCHAR, the user-defined VARRAY type must be twice as big as the length specified for the table column.
- The number of rows in the VARRAY that are sent by the MobiLink server to the Oracle consolidated database is set by the -s option, not the size of the VARRAY declared in the VARRAY type. The -s option must not be bigger than the smallest VARRAY type size in use by synchronization scripts. If it is bigger, the MobiLink server issues an error. See "-s mlsrv12 option" on page 64.

SQL Anywhere consolidated database

Permissions

Before running the setup script, you should be aware of the following requirements:

- The database user who runs the setup script is expected to be the same one used to update the MobiLink system tables during synchronization. This user must be used to start the MobiLink server and to configure MobiLink applications. See "Required permissions" on page 22.
- The RDBMS user that the MobiLink server uses to connect to the consolidated database must be able to use the MobiLink system tables, procedures, and so on, without any qualifiers (for example, SELECT * from ml_user). See "MobiLink server system tables" on page 4.

Setting up SQL Anywhere as a consolidated database

To set up SQL Anywhere to work as a MobiLink consolidated database, you must run a setup procedure that adds MobiLink system tables, stored procedures, triggers, and views that are required for MobiLink synchronization. There are multiple ways you can do this:

- Run the *syncsa.sql* setup script, located in *install-dir\MobiLink\setup*.
- In the MobiLink 12 plug-in for Sybase Central, choose **Folders** from the **View** menu. Open your MobiLink project and expand **Consolidated Databases**. Right-click the database name and choose **Check MobiLink System Setup**. If your database requires setup, you are prompted to continue.
- When you use the **Create Synchronization Model Wizard** or **Deploy Synchronization Model Wizard**, the system setup is checked when you connect to your server database. If your database requires setup, you are prompted to continue.

Setting up the ODBC driver

You must set up an ODBC DSN for your SQL Anywhere consolidated database. The ODBC driver for SQL Anywhere is installed with SQL Anywhere.

For information about the SQL Anywhere ODBC driver, see "Creating ODBC data sources" [*SQL Anywhere Server - Database Administration*].

Isolation level

See "MobiLink isolation levels" on page 119.

MobiLink server

Running the MobiLink server

All MobiLink clients synchronize through the MobiLink server. None connect directly to a database server. You must start the MobiLink server before a MobiLink client synchronizes.

For a list of mlsrv12 command line options, see "MobiLink server options" on page 32.

The MobiLink server opens database connections, via ODBC, with your consolidated database server. It then accepts connections from remote applications and controls the synchronization process.

To start the MobiLink server

- Run mlsrv12. Use the -c option to specify the ODBC connection parameters for your consolidated database.

For information about connection parameters, see "-c mlsrv12 option" on page 39.

You must specify connection parameters. Other options are available, but are optional. These options allow you to specify how the server works. For example, you can specify a cache size and logging options.

For more information about mlsrv12 options, see "mlsrv12 syntax" on page 32.

Note
The mlsrv12 options allow you to specify how the MobiLink server works. To control what the server does during synchronization, you define scripts that are invoked at synchronization events. See "Synchronization events" on page 297.

Example

The following command starts the MobiLink server using the ODBC data source *SQL Anywhere 12 CustDB* to identify the consolidated database. Enter the entire command on one line.

```
mlsrv12
  -c "dsn=SQL Anywhere 12 CustDB;uid=DBA;pwd=sql"
  -zs MyServer
  -o mlsrv.log
  -vcr
  -x tcpip(port=3303)
```

In this example, the -c option provides a connection string that contains an ODBC data source name (DSN) and authentication. The -zs option provides a server name. The -o option specifies that the log file should be named *mlsrv.log*. The contents of *mlsrv.log* are verbose because of the -vcr option. The -x option opens a port for version 10 and later clients.

For more information about the options described in the previous example, see:

- "-c mlsrv12 option" on page 39
- "-zs mlsrv12 option" on page 82
- "-o mlsrv12 option" on page 53
- "-v mlsrv12 option" on page 70
- "-x mlsrv12 option" on page 75

You can also start the MobiLink server as a Windows service or Unix daemon. See "Running the MobiLink server outside the current session" on page 24.

Required permissions

You must specify a database user for the MobiLink server to connect to the database server. You specify the database user with the mlsrv12 -c option or in the ODBC data source. See "-c mlsrv12 option" on page 39.

This database user must have full select, insert, update, and delete permissions on the MobiLink system tables, and must also have execute permissions on the MobiLink system procedures. By default, the database user who runs the MobiLink setup script has these permissions. If you want to use another database user to run the MobiLink server, you must grant permissions for that user on the ml_* tables and the ml_add_*_script system procedures.

For example, in a SQL Anywhere consolidated database you can grant the required permissions as follows:

```
CREATE USER DBUser IDENTIFIED BY SQL;
GRANT ALL ON dbo.ml_user to DBUser;
...
GRANT EXECUTE ON dbo.ml_add_table_script TO DBUser;
...
```

For an example of code that grants the necessary permissions to all MobiLink system objects, see "Upgrading your consolidated database" [*SQL Anywhere 12 - Changes and Upgrading*].

For a list of all MobiLink system procedures, see "MobiLink server system procedures" on page 691.

The database user also needs the appropriate permission on all tables referenced in the MobiLink scripts, ands execute permissions on any procedures referenced in the MobiLink scripts.

Some types of consolidated databases require the database user used by MobiLink server to have specific permissions against a small number of system tables and/or views. See the following topics for information about specific consolidated databases:

- "Adaptive Server Enterprise consolidated database" on page 9
- "IBM DB2 LUW consolidated database" on page 10
- "Microsoft SQL Server consolidated database" on page 13
- "MySQL consolidated database" on page 14
- "Oracle consolidated database" on page 17

For more information about setup scripts, see "Setting up a consolidated database" on page 2.

Stopping the MobiLink server

The MobiLink server is stopped from the computer where the server was started. You can stop the MobiLink server in the following ways:

- Use the MobiLink stop utility (mlstop). See "MobiLink stop utility (mlstop)" on page 731.
- Click **Shut down** on the MobiLink server messages window.
- In Windows, right-click the MobiLink server icon in the system tray and choose **Shut down**.

- When running on Unix without the MobiLink server messages window, type Q.
- Use the shutdown method in the MobiLink server API.

See also

- Server API for Java: “shutdown method” on page 567
- Server API for .NET: “Shutdown method” on page 651

Logging MobiLink server actions

Logging the actions that the server takes is particularly useful during the development process and when troubleshooting. Verbose output is not recommended for normal operation of a production environment because it can slow performance.

Logging output to a file

Selected logging output is sent to the MobiLink server messages window. In addition, you can send the output to a message log file using the -o option. The following command sends output to a message log file named *mlsrv.log*.

```
mlsrv12 -o mlsrv.log -c ...
```

You can control the size of log files, and specify what you want done when a file reaches its maximum size with the following options:

- Use the -o option to specify that a log file should be used.
- Use the -ot option to specify that a log file should be used and you want the previous contents of the file to be deleted before messages are sent to it.
- In addition to -o or -ot, use the -on option to specify the size at which the log file is renamed with the extension .old and a new file is started with the original name. This option limits the total disk space taken up by the message log files.
- In addition to -o or -ot, use the -os option to specify the size of old log files when they are assigned a new name based on the date and a sequential number.

See:

- “-o mlsrv12 option” on page 53
- “-on mlsrv12 option” on page 54
- “-os mlsrv12 option” on page 55
- “-ot mlsrv12 option” on page 56

Controlling the amount of logging output

You can control what information is logged to the message log file and displayed in the MobiLink server messages window using the -v option. See “-v mlsrv12 option” on page 70.

Controlling which warning messages are reported

You can control which warning messages are reported.

For more information, see:

- "-zw mlsrv12 option" on page 84
- "-zwd mlsrv12 option" on page 85
- "-zwe mlsrv12 option" on page 85

Viewing MobiLink server logs

You can view MobiLink logs in the following ways:

- In the MobiLink server messages window
- By opening the log file
- Using the MobiLink server Log File Viewer in Sybase Central

To view log information outside the MobiLink server messages window, you must log the information to a file. See "Logging output to a file" on page 23.

MobiLink server Log File Viewer

To view MobiLink server logs, open Sybase Central and choose **Tools** » **MobiLink 12** » **MobiLink server Log File Viewer**. You are prompted to choose a log file to view. By holding down the Shift key, you can open multiple log files at the same time.

The MobiLink server Log File Viewer reads information that is stored in MobiLink log files. It does not connect to the MobiLink server or change the composition of log files.

The MobiLink server Log File Viewer allows you to filter the information that you view. In addition, it provides statistics based on the information in the log.

Running the MobiLink server outside the current session

You can set up the MobiLink server to be available all the time. To make this easier, you can run the MobiLink server for Windows and for Unix so that it remains running when you log off the computer. The way to do this depends on your operating system.

- **Unix daemon** You can run the MobiLink server as a daemon using the mlsrv12 -ud option, enabling the MobiLink server to run in the background, and to continue running after you log off. See "-ud mlsrv12 option" on page 69.
- **Windows service** You can run the Windows MobiLink server as a service. See "Running the Windows MobiLink server as a service" on page 25.

To stop a MobiLink server that is running as a service, you can use mlstop, dbsvc, or the Windows Service Manager.

See also

- "Service utility (dbsvc) for Linux" [*SQL Anywhere Server - Database Administration*]
- "Running the database server outside the current session" [*SQL Anywhere Server - Database Administration*]

Running the Unix MobiLink server as a daemon

To run the MobiLink server in the background on Unix, and to enable it to run independently of the current session, you run it as a daemon.

To run the Unix MobiLink server as a daemon

- Use the -ud option when starting the MobiLink server. For example:

```
mlsrv12 -c "dsn=SQL Anywhere 12 Demo;uid=DBA;pwd=sql" -ud
```

See "-ud mlsrv12 option" on page 69.

See also

- "Service utility (dbsvc) for Linux" [*SQL Anywhere Server - Database Administration*]

Running the Windows MobiLink server as a service

To run the Windows MobiLink server in the background, and to enable it to run independently of the current session, you run it as a service.

You can run the following service management tasks from the command line, or on the **Services** tab in Sybase Central:

- Add, edit, and remove services.
- Start and stop services.
- Modify the parameters governing a service.

See also

- "Service utility (dbsvc) for Windows" [*SQL Anywhere Server - Database Administration*]

Adding, modifying, and deleting services

The service icons in Sybase Central display the current state of each service using an icon that indicates whether the service is running or stopped.

To add a new service (Sybase Central)

1. In Sybase Central, in the left pane, click MobiLink 12.
2. In the right pane, click the **Services** tab.
3. In the right pane, right-click and choose **New** » **Service**.
4. Follow the instructions in the **Create Service Wizard**.

You can also use the dbsvc utility to create the service. See "Service utility (dbsvc) for Windows" [*SQL Anywhere Server - Database Administration*].

To delete a service (Sybase Central)

- Choose the service and then click **Edit** » **Delete**.

To change the parameters for a service

- Right-click the service and choose **Properties**.

Changes to a service configuration take effect the next time the service is started.

Setting the startup option

The following options govern startup behavior for MobiLink services. You can set them on the **General** tab of the **Service Properties** window.

- **Automatic** If you choose **Automatic**, the service starts whenever the Windows operating system starts. This setting is appropriate for database servers and other applications running all the time.
- **Manual** If you choose **Manual**, the service starts only when a user with Administrator permissions starts it. For information about Administrator permissions, see your Windows documentation.
- **Disabled** If you choose **Disabled**, the service does not start.

The startup option is applied the next time Windows is started.

Specifying command line options

The **Configuration** tab of the **Service Properties** window provides a **File name** text box for entering the program executable path and a **Parameters** text box for entering command line options for a service. Do not type the name of the program executable in the **Parameters** box.

For example, to start a MobiLink synchronization service with verbose logging, type the following in the **Parameters** box:

```
-c "dsn=SQL Anywhere 12 Demo;uid=DBA;pwd=sql"
-vc
```

The command line options for a service are the same as those for the executable. See "MobiLink server options" on page 32.

Setting account options

You can choose which account the service runs under. Most services run under the special LocalSystem account, which is the default option for services. You can set the service to log on under another account by opening the **Account** tab on the **Service Properties** window, and entering the account information.

If you choose to run the service under an account other than LocalSystem, that account must have the "log on as a service" privilege. For information about advanced privileges, see your Microsoft Windows documentation.

Whether an icon for the service appears on the taskbar or desktop is dependent on the account you select, and whether **Allow Service To Interact with Desktop** is checked, as follows:

- If a service runs under LocalSystem, and **Allow Service To Interact with Desktop** is checked in the **Service Properties** window, an icon appears on the desktop of every user logged in to Windows XP/ 200x on the computer running the service. Any user can open the application window and stop the program running as a service.
- If a service runs under LocalSystem, and **Allow Service To Interact with Desktop** is unchecked in the **Service Properties** window, no icon appears on the desktop for any user. Only users with permissions to change the state of services can stop the service.
- If a service runs under another account, no icon appears on the desktop. Only users with permissions to change the state of services can stop the service.

Changing the executable file

To change the program executable file associated with a service in Sybase Central, click the **Configuration** tab on the **Service Properties** window and type the new path and file name in the **File Name** box.

If you move an executable file to a new directory, you must modify this entry.

Starting and stopping

To start or stop a service

1. In Sybase Central, click MobiLink 12 in the left pane, and then open the **Services** tab in the right pane.
2. Right-click the service and choose **Start** or **Stop**.

If you start a service, it keeps running until you stop it. Closing Sybase Central or logging off does not stop the service.

Stopping a service closes all network connections and stops MobiLink server. For other applications, the program closes down.

Running more than one service at a time

Although you can use the Windows Service Manager in the Control Panel for some tasks, you cannot install or configure a MobiLink service from the Windows Service Manager. You can use Sybase Central to perform all the service management for MobiLink.

When you open the Windows Service Manager from the Windows Control Panel, a list of services appears. The names of the SQL Anywhere services are formed from the Service Name you provided when installing the service, prefixed by SQL Anywhere. All the installed services appear together in the list.

This section describes topics specific to running more than one service at a time.

Service dependencies

In some circumstances you may want to run more than one executable as a service, and these executables may depend on each other. For example, you must run the MobiLink server and the consolidated database server to synchronize.

Services must start in the correct order. If a MobiLink synchronization service starts before the consolidated database server has started, MobiLink fails because it cannot find the consolidated database server. The sequence must be such that the database server is running when you start the MobiLink server. (This does not apply if the consolidated database server is on another computer.)

You can prevent these problems using service groups, which you manage from Sybase Central.

Service groups

You can assign each service on your system to be a member of a service group. By default, each service belongs to a group. The default group for the MobiLink server is SQLANYMobiLink.

Before you can configure your services to ensure they start in the correct order, you must check that your service is a member of an appropriate group. You can check which group a service belongs to, and change this group, from Sybase Central.

To check and change which group a service belongs to

1. In Sybase Central, click MobiLink 12 in the left pane, and then open the **Services** tab in the right pane.
2. Right-click the service and choose **Properties**.
3. Click the **Dependencies** tab. The top text box displays the name of the group the service belongs to.
4. Click **Change** to display a list of available groups on your system.
5. Select one of the groups, or type a name for a new group.
6. Click **OK** to assign the service to that group.

Managing service dependencies

With Sybase Central, you can specify dependencies for a service. For example:

- You can ensure that at least one group has started before the current service.
- You can ensure that any service starts before the current service.

To add a service or group to a list of dependencies

1. In Sybase Central, click MobiLink 12 in the left pane, and then open the **Services** tab in the right pane.
2. Right-click the service and choose **Properties**.
3. Click the **Dependencies** tab.
4. Click **Add Services** or **Add Service Groups** to add a service or group to the list of dependencies.
5. Select one of the services or groups from the list.
6. Click **OK** to add the service or group to the list of dependencies.

Running the MobiLink server in a server farm

Separately licensed component required
Running the MobiLink server in a server farm is a feature of the MobiLink high availability option, which requires a separate license. See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

A MobiLink server farm is an environment where there is more than one MobiLink server synchronizing the same set of remote databases with the same consolidated database. This is often required for large scale deployments or for fail-over capability. These MobiLink server farm deployments may require the use of the Relay Server if an HTTP communication link is used. For TCP-based streams, a TCP load balancer should work. When using multiple servers, restartable download does not work.

To enable shared server state

- If you are using the Notifier with server-initiated synchronization, use the -lsc option to specify the local server connection settings. These settings are passed to the other servers in the farm so that they can connect to each other to share the handling of notifications. For example, if running on host **farm_host22**: `mlsrv12 -x tcpip(port=3245) -zs server5 -lsc tcpip(host=farm_host22;port=3245) -c ...`

MobiLink arbiter server

When running a MobiLink server farm with server-initiated synchronization or QAnywhere messaging, use the MobiLink arbiter server to make sure there is always one primary server in the farm. Having a primary server at all times prevents redundant notifications and lost messages.

Use the **mlarbiter** command to start the MobiLink arbiter and use the MobiLink server -ca option along with the -lsc option to start the MobiLink servers with the arbiter information. See "MobiLink arbiter server utility (mlarbiter)" on page 736 and "-ca mlsrv12 option" on page 40.

See also

- "-lsc mlsrv12 option" on page 51
- "-ca mlsrv12 option" on page 40

Memory use in the MobiLink server

The MobiLink server primarily uses memory in the following ways:

- caching
- for VMs
- storing system data

Cache

In a typical MobiLink server, most of the memory is used by the memory cache, which stores row data and related data structures and network buffers. In general, data that is unbounded in either size or quantity is stored in the cache. If the amount of data to store in the memory cache exceeds the cache's size the excess is transferred to disk, which can become a potential bottleneck in synchronization performance. The degree to which swapping to disk is a problem depends on the amount and frequency of disk I/O and the speed of the disk. To eliminate this potential bottleneck, it is recommended that you avoid swapping data to disk completely. Watch for warning 10082 in the MobiLink server log, or the "Cache is full" alert in the SQL Anywhere Monitor.

By default, the MobiLink server automatically grows its cache up to 60% of the available process address space and shrinks its cache if other processes on the system require more memory, or the server's non-cache memory grows larger. The initial maximum and minimum cache sizes can be controlled with the -cinit, -cmax and -cmin mlsrv12 server options. If desired, the server can disable dynamic cache sizing by specifying the same value for the -cmax and -cmin option values.

See:

- “-cmax mlsrv12 option” on page 42
- “-cmin mlsrv12 option” on page 42
- “-cinit mlsrv12 option” on page 41

There are several ways to find out information about the cache:

- **Server Cache Size** This is a SQL Anywhere Monitor metric that gives the current cache size. See “List of metrics” on page 209.
- **Percent of Pages Locked** This is a SQL Anywhere Monitor metric that gives the percentage of pages in the cache that can not be transferred to disk. See “List of metrics” on page 209.
- **Percent of Pages Used** This is a SQL Anywhere Monitor metric that gives the percentage of pages that contain valid data. See “List of metrics” on page 209.
- **PAGES_LOCKED_MAX** The -ppv option for mlsrv12 can print the number of pages in the memory cache. See “-ppv mlsrv12 option” on page 56.
- **PAGES_LOCKED** The -ppv option for mlsrv12 can print the number of cache pages loaded into memory. See “-ppv mlsrv12 option” on page 56.

- **PAGES_USED** The -ppv option for mlsrv12 can print the number of cache pages used. See "-ppv mlsrv12 option" on page 56.

- **-vk option for mlsrv12** Causes the server to print a line to the log when the cache grows or shrinks. See "-v mlsrv12 option" on page 70.

VMs

Another major use of memory within the MobiLink server is the embedded Java and .NET VMs. In deployments that make heavy use of the direct row API or that embed application servers like JBoss or Tomcat, the VM memory use can exceed the memory cache.

When using a Java VM, you usually have some control over the amount of memory the VM uses. Most Java VMs provide the -Xms and -Xmx switches that allow you to specify the maximum and initial VM heap size, respectively.

You can find out the amount of memory being used by the embedded VMs with the **VM Memory Usage** SQL Anywhere Monitor metric, or with the **VM_MEM_USE** -ppv value.

See:

- "List of metrics" on page 209
- "-ppv mlsrv12 option" on page 56

Other memory usage

The remaining memory is used for storing state information about each synchronization, thread stacks, inter-thread communication and the ODBC driver. Typically, this accounts for a few tens of megabytes of memory and the amount used is bounded by the -nc, -sm and -w options for mlsrv12. It is worth noting that non-SQL Anywhere ODBC drivers can use a significant amount of memory, particularly when processing BLOB columns.

See:

- "-nc mlsrv12 option" on page 52
- "-sm mlsrv12 option" on page 67
- "-w mlsrv12 option" on page 74

Troubleshooting MobiLink server startup

This section describes some common problems when starting the MobiLink server.

Ensure that network communication software is running

Appropriate network communication software must be installed and running before you run the MobiLink server. If you are running reliable network software with just one network installed, this should be straightforward. You should confirm that other software requiring network communications is working properly before running the MobiLink server.

You may want to confirm that ping and telnet are working properly. The ping and telnet applications are provided with many TCP/IP protocol stacks.

Debug network communications startup problems

If you are having problems establishing a connection across a network, you can use debugging options at both the client and server to diagnose problems. The startup information appears in the MobiLink server messages window: you can use the -o option to log the results to an output file.

See "Logging MobiLink server actions" on page 23.

MobiLink server options

mlsrv12 syntax

Starts a MobiLink server.

Syntax

mlsrv12 -c "*connection-string***"** [*options*]

Option	Description
@data	Read in options from the specified environment variable or configuration file. See "@data mlsrv12 option" on page 37.
-a	Keep using a consolidated database connection after a synchronization error on that connection. See "-a mlsrv12 option" on page 38.
-b	Trim blank padding of strings. See "-b mlsrv12 option" on page 38.
-bn *size*	Specify the maximum number of bytes to consider when comparing BLOBs for conflict detection. See "-bn mlsrv12 option" on page 39.
-c *"keyword=value; ..."*	Supply ODBC database connection parameters for your consolidated database. This option is required. See "-c mlsrv12 option" on page 39.
-ca *host_or_ip*	Set the host name or IP address for the MobiLink arbiter server. See "-ca mlsrv12 option" on page 40.

Option	Description
-cinit *size*	Specify the initial size for the server memory cache. See "-cinit mlsrv12 option" on page 41.
-cm *size*	Specify the server memory cache size. See "-cm mlsrv12 option" on page 41.
-cmax *size*	Specify the maximum size for the server memory cache. See "-cmax mlsrv12 option" on page 42.
-cmin *size*	Specify the minimum size for the server memory cache. See "-cmin mlsrv12 option" on page 42.
-cn *connections*	Set the maximum number of simultaneous connections with the consolidated database server. See "-cn mlsrv12 option" on page 43.
-cr *count*	Set the maximum number of database connection retries. See "-cr mlsrv12 option" on page 44.
-cs *"keyword=value; ..."*	Supply system database connection parameters for your MobiLink System Database (MLSD). See "-cs mlsrv12 option" on page 44.
-ct *connection-timeout*	Set the length of time a connection may be unused before it is timed out. See "-ct mlsrv12 option" on page 45.
-dl	Display all log messages in the MobiLink server messages window. See "-dl mlsrv12 option" on page 45.
-dr	For Adaptive Server Enterprise only. Ensures that tables involved in synchronization do not use the DataRow locking scheme. See "-dr mlsrv12 option" on page 45.
-ds *size*	Specify the maximum amount of data that can be stored for use in all restartable downloads. See "-ds mlsrv12 option" on page 46.
-dsd	Disable snapshot isolation, which is the default download isolation level for SQL Anywhere and Microsoft SQL Server consolidated databases. See "-dsd mlsrv12 option" on page 46.
-dt	Detect transactions only within the current database. See "-dt mlsrv12 option" on page 47.

Option	Description
-e *filename*	Store remote error logs sent into the named file. See "-e mlsrv12 option" on page 48.
-esu	Use snapshot isolation for uploads. See "-esu mlsrv12 option" on page 48.
-et *filename*	Store remote error logs sent into the named file, but delete the contents first if the file exists. See "-et mlsrv12 option" on page 49.
-fips	Forces all secure MobiLink streams to be FIPS-compliant. See "-fips mlsrv12 option" on page 49.
-ftr *path*	Specifies a location for files that are to be downloaded by the MobiLink transfer utility (mlfiletransfer). See "-ftr mlsrv12 option" on page 50.
-ftru	Specifies a location for files uploaded with the MobiLink File Transfer utility (mlfiletransfer). See "-ftru mlsrv12 option" on page 50.
-lsc *protocol*[*protocol-options*]	Specifies the local server connect information. See "-lsc mlsrv12 option" on page 51.
-m [*filename*]	Enables QAnywhere messaging. See "-m mlsrv12 option" on page 52.
-nc *connections*	Sets maximum number of concurrent network connections. See "-nc mlsrv12 option" on page 52.
-notifier	Starts a Notifier for server-initiated synchronization. See "-notifier mlsrv12 option" on page 53.
-o *logfile*	Log messages to a file. See "-o mlsrv12 option" on page 53.
-on *size*	Set maximum size for log file. See "-on mlsrv12 option" on page 54.
-oq	Prevent the popup window on startup error. See "-oq mlsrv12 option" on page 55.
-os *size*	Maximum size of old log files. See "-os mlsrv12 option" on page 55.
-ot *logfile*	Log messages to a file, but delete its contents first. See "-ot mlsrv12 option" on page 56.

Option	Description
-ppv *period*	Causes MobiLink to print new periodic monitoring values according to the period specified. See "-ppv mlsrv12 option" on page 56.
-q	Minimize the MobiLink server messages window. See "-q mlsrv12 option" on page 61.
-r *retries*	Retry deadlocked uploads at most this many times. See "-r mlsrv12 option" on page 61.
-rd *delay*	Set maximum delay, in seconds, before retrying a deadlocked transaction. See "-rd mlsrv12 option" on page 62.
-rp *directory*	Specifies the directory to which synchronizations are recorded for playback with the mlreplay utility. See "-rp mlsrv12 option" on page 63.
-rrp *directory*	Causes the MobiLink server to run the mlreplay utility in the given directory when the server starts. See "-rrp mlsrv12 option" on page 62.
-s *count*	Specify the maximum number of rows to be uploaded to or fetched from the consolidated database at once. See "-s mlsrv12 option" on page 64.
-sl dnet *script-options*	Set the .NET Common Language Runtime (CLR) options and force the CLR to load on startup. See "-sl dnet mlsrv12 option" on page 64.
-sl java *script-options*	Set the Java VM options and force loading of the Java VM on startup. See "-sl java mlsrv12 option" on page 65.
-sm *number*	Set the maximum number of synchronizations that can be worked on concurrently. See "-sm mlsrv12 option" on page 67.
-tc *minutes*	Set the timeout threshold for SQL script execution. See "-tc mlsrv12 option" on page 68.
-tf	Fail the SQL script execution when the timeout threshold is reached (not for Oracle). See "-tf mlsrv12 option" on page 68.

Option	Description
-tx *count*	For transactional uploads, batches groups of transactions and commits them together. See "-tx mlsrv12 option" on page 69.
-ud	On Unix platforms, run as a daemon. See "-ud mlsrv12 option" on page 69.
-ui	For Linux with X windows, starts the MobiLink server in shell mode if a usable display isn't available. See "-ui mlsrv12 option" on page 69.
-ux	For Linux with X windows, opens the MobiLink server messages window. See "-ux mlsrv12 option" on page 70.
-v [*levels*]	Controls the type of messages written to the log file. See "-v mlsrv12 option" on page 70.
-w *count*	Set the number of database worker threads. See "-w mlsrv12 option" on page 74.
-wu *count*	Set the maximum number of database worker threads permitted to process uploads concurrently. See "-wu mlsrv12 option" on page 75.
-x *protocol*[**(***network-parameters***)**]	Specify the communications protocol. Optionally, specify network parameters in form *parameter=value*, with multiple parameters separated by semicolons. See "-x mlsrv12 option" on page 75.
-zf	Specifies that the MobiLink server should check for script changes at the beginning of each synchronization. See "-zf mlsrv12 option" on page 82.
-zp	Ignore some apparent differences between timestamp values when the remote and consolidated databases have different precision. See "-zp mlsrv12 option" on page 82.
-zs *name*	Specify a server name. See "-zs mlsrv12 option" on page 82.
-zt *number*	Specify the maximum number of processors used to run the MobiLink server. See "-zt mlsrv12 option" on page 83.

Option	Description
-zu { + | - }	Controls the automatic addition of users when the authenticate_user script is undefined. See "-zu mlsrv12 option" on page 83.
-zus	Causes MobiLink to invoke upload scripts for tables for which there is no upload. See "-zus mlsrv12 option" on page 84.
-zw 1,...5	Controls which levels of warning message to display. See "-zw mlsrv12 option" on page 84.
-zwd *code*	Disables specific warning codes. See "-zwd mlsrv12 option" on page 85.
-zwe *code*	Enables specific warning codes. See "-zwe mlsrv12 option" on page 85.

Remarks

The MobiLink server opens connections, via ODBC, with your consolidated database server. It then accepts connections from client applications and controls the synchronization process.

You must supply connection parameters for the consolidated database using the -c option. The command line options may be specified in any order. The -c option is shown here as the first item in a command string as a convention only. It can be anywhere in a list of options, but must precede a connection string.

Unless your ODBC data source is configured to automatically start the consolidated database, the database must be running before you start the MobiLink server.

See also

- "MobiLink server" on page 20

@data mlsrv12 option

Reads in options from the specified environment variable or configuration file.

Syntax

mlsrv12 -c "*connection-string***" @***data* ...

Remarks

Use this option to read in mlsrv12 command line options from the specified environment variable or configuration file. If both exist with the same name that is specified, the environment variable is used.

For more information about configuration files, 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*].

-a mlsrv12 option

Instructs the MobiLink server to keep using a consolidated database connection after a synchronization error on that connection.

Syntax

mlsrv12 -c "*connection-string***" -a** ...

Remarks

By default, if an error occur during synchronization the MobiLink server automatically disconnects from the consolidated database, and then re-establishes the connection. Reconnecting ensures that the following synchronization starts from a known state. When this behavior is not required, you can use this option to disable it. The maintenance of state information depends on your requirements and may vary depending on the ways in which you configure MobiLink scripting to work with the RDBMS. This applies even if that database is an Oracle, SQL Anywhere database, or other supported product. Some status information may need to be re-initialized depending on the client application.

-b mlsrv12 option

For columns of type VARCHAR, CHAR, LONG VARCHAR, or LONG CHAR, removes trailing blanks from strings during synchronization.

Syntax

mlsrv12 -c "*connection-string***" -b** ...

Remarks

Note
It is recommended that you use VARCHAR in the consolidated database rather than CHAR, so that this problem does not occur.

This option helps resolve differences between the SQL Anywhere CHAR data type and the CHAR or VARCHAR data type used by the consolidated database. The SQL Anywhere CHAR data type is equivalent to VARCHAR. However, in most consolidated databases that are not SQL Anywhere, the CHAR(n) data type is blank-padded to n characters.

When -b is specified, the MobiLink server removes trailing blanks from strings for columns of type CHAR, VARCHAR, LONG CHAR, or LONG VARCHAR if the column on the remote is a string. The trimmed data is then downloaded to the remote databases.

This option can also be used to properly detect conflict updates when the upload_fetch or upload_fetch_column_conflict script is used. For each upload update row, the MobiLink server fetches the row from the consolidated database for the given primary key, compares the row with the pre-image of the update, and then determines whether the update is a conflict update. When -b is used, MobiLink trims trailing blanks from columns of type CHAR, VARCHAR, LONG CHAR, or LONG VARCHAR before doing the comparison.

See also

- "CHAR columns" on page 5
- "NVARCHAR data type" [*SQL Anywhere Server - SQL Reference*]
- "upload_fetch table event" on page 454
- "upload_fetch_column_conflict table event" on page 469

Example

If the -b option is not used, a primary-key value of 'abc' uploaded from a SQL Anywhere or UltraLite remote to a CHAR(10) column in the consolidated database becomes 'abc' followed by seven blank spaces. If the same row is downloaded, then it appears on the remote as 'abc' followed by seven spaces. If the remote database is not blank-padded, then the remote contains two rows: both 'abc' and 'abc' followed by seven spaces. There is now a duplicate row on the remote.

If the -b option is used, a primary-key value of 'abc' uploaded from a SQL Anywhere or UltraLite remote to a CHAR(10) column in the consolidated database becomes 'abc' followed by seven spaces. Seven spaces still pad the value to ten characters, but if the same row is downloaded, then MobiLink server strips the trailing spaces, and the value appears on the remote as 'abc'. The -b option fixes the duplicate row problem.

-bn mlsrv12 option

Sets the maximum number of BLOB bytes to compare during conflict detection.

Syntax

mlsrv12 -c "*connection-string***" -bn** *size* ...

Remarks

When two BLOBs contain similar or identical values, the operation of comparing them for filtering or conflict detection can be expensive due to the amount of data involved. This option tells the MobiLink server to consider only the first *size* bytes of two BLOBs when making the comparison. The default is to compare the two BLOBs in their entirety.

Under some situations, limiting the maximum amount of data compared can speed synchronization substantially; however, it can also cause errors. For example, if two large BLOBs differ only in the last few bytes, the MobiLink server may consider them identical when in fact they are not.

-c mlsrv12 option

Specifies connection parameters for the consolidated database.

Syntax

mlsrv12 -c "*connection-string***"** ...

Remarks

The connection string must give the MobiLink server enough information to connect to the consolidated database. The connection string is required.

The connection string must specify connection parameters in the form *keyword=value*, separated by semicolons, with no spaces between parameters.

Connection parameters must be included in the ODBC data source specification if not given in the command line. Check your RDBMS and ODBC data source to determine required connection data.

For a complete list of SQL Anywhere connection parameters, see "Connection parameters" [*SQL Anywhere Server - Database Administration*].

For information about how to hide the password, see "File Hiding utility (dbfhide)" [*SQL Anywhere Server - Database Administration*].

Example

```
mlsrv12 -c "dsn=odbcname;uid=DBA;pwd=sql"
```

-ca mlsrv12 option

Sets the MobiLink arbiter server's host name or IP address to let the MobiLink server know where the MobiLink arbiter is running.

Syntax

mlsrv12 -c "*connection-string***" -ca** *host_or_ip* ...

Remarks

All of the MobiLink servers in the same server farm must contain the same setting for the -ca option.

Along with the -ca option, also use the -lsc option to specify the connection string for the local MobiLink server. See "-lsc mlsrv12 option" on page 51.

The -ca and -lsc command line options are ignored by the MobiLink server if its command line does not contain either -notifier or -m.

> **Note**
> Port 4953 has been assigned to the MobiLink arbiter so this port number cannot be used by any other applications on the machine where the MobiLink arbiter server is running.

See also

- "-lsc mlsrv12 option" on page 51
- "-notifier mlsrv12 option" on page 53
- "-m mlsrv12 option" on page 52

-cinit mlsrv12 option

Sets the initial size for the server memory cache.

Syntax

mlsrv12 -c "*connection-string***" -cinit** *size*[**k** | **m** | **g** | **p**] ...

Remarks

The initial amount of memory the server uses for holding table data, network buffers, cached download data, and other structures used for synchronization. When the server has more data than can be held in this memory pool, the data is swapped to disk. Swapping data to disk results in a significant performance penalty.

The *size* is the amount of memory to reserve in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively. If no letter follows the number, the size is in bytes.

The unit **p** is a percentage either of the physical system memory, or of the process addressable space, whichever is lower. The maximum process addressable space depends on the operating system. For example:

- 2.5 GB for Windows 32-bit Advanced Server, Enterprise Server and Datacenter Server
- 3.5 GB for the 32-bit database server running on Windows x64 Edition
- 1.5 GB on all other 32-bit systems
- On 64-bit database servers, the cache size can be considered unlimited

The default is **50m**.

See also

- "-cmax mlsrv12 option" on page 42
- "-cmin mlsrv12 option" on page 42
- "Memory use in the MobiLink server" on page 30

-cm mlsrv12 option

Sets the maximum size for the server memory cache. Note that this option is an alias for the -cmax option. See "-cmax mlsrv12 option" on page 42.

-cmax mlsrv12 option

Sets the maximum size for the server memory cache.

Syntax

mlsrv12 -c "*connection-string***" -cmax** *size*[**k** | **m** | **g** | **p**] ...

Remarks

The maximum amount of memory the server uses for holding table data, network buffers, cached download data, and other structures used for synchronization. When the server has more data than can be held in this memory pool, the data is stored swapped to disk. Swapping data to disk results in a significant performance penalty.

The *size* is the amount of memory to reserve in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively. If no letter follows the number, the size is in bytes.

The unit **p** is a percentage either of the physical system memory, or of the process addressable space, whichever is lower. The maximum process addressable space depends on the operating system. For example:

- 2.5 GB for Windows 32-bit Advanced Server, Enterprise Server and Datacenter Server
- 3.5 GB for the 32-bit database server running on Windows x64 Edition
- 1.5 GB on all other 32-bit systems
- On 64-bit database servers, the cache size can be considered unlimited

The default is **70p**.

See also

- "-cinit mlsrv12 option" on page 41
- "-cmin mlsrv12 option" on page 42
- "Memory use in the MobiLink server" on page 30

-cmin mlsrv12 option

Sets the minimum size for the server memory cache.

Syntax

mlsrv12 -c "*connection-string***" -cmin** *size*[**k** | **m** | **g** | **p**] ...

Remarks

The minimum amount of memory the server uses for holding table data, network buffers, cached download data, and other structures used for synchronization. When the server has more data than can be held in this memory pool, the data is swapped to disk. Swapping data to disk results in a significant performance penalty.

The *size* is the amount of memory to reserve in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively. If no letter follows the number, the size is in bytes.

The unit **p** is a percentage either of the physical system memory, or of the process addressable space, whichever is lower. The maximum process addressable space depends on the operating system. For example:

- 2.5 GB for Windows 32-bit Advanced Server, Enterprise Server and Datacenter Server
- 3.5 GB for the 32-bit database server running on Windows x64 Edition
- 1.5 GB on all other 32-bit systems
- On 64-bit database servers, the cache size can be considered unlimited

The default is **50m**.

See also

- "-cmax mlsrv12 option" on page 42
- "-cinit mlsrv12 option" on page 41
- "Memory use in the MobiLink server" on page 30

-cn mlsrv12 option

Sets the maximum number of simultaneous consolidated database connections for database worker threads.

Syntax

mlsrv12 -c "*connection-string***" -cn** *value* ...

Remarks

Specifies the maximum number of simultaneous connections that the MobiLink server should make to the consolidated database for database worker threads. The minimum and the default value are the number of database worker threads. A warning is issued if the supplied value is smaller than the number of the database worker threads, and the value is automatically adjusted upward.

This type of MobiLink database connection is only used for synchronizations using one script version. When the MobiLink server is using all the database connections that it is permitted by the -cn option, if a synchronization is pending but no database connection for its script version currently exists, the MobiLink server disconnects a connection and then creates a new database connection for the pending synchronization's script version.

A value larger than the number of database worker threads may speed performance, particularly if connecting to the consolidated database is slow or if multiple script versions are in use. The optimum maximum number of database connections is the number of script versions times the number of database worker threads. Connections above this optimum value do not necessarily speed synchronization, and needlessly consume resources in both the MobiLink server and the consolidated database server.

See also

- "-w mlsrv12 option" on page 74

-cr mlsrv12 option

Sets the maximum number of database connection retries.

Syntax

mlsrv12 -c "*connection-string***" -cr** *value* …

Remarks

Set the maximum number of times that the MobiLink server attempts to connect to the database, before quitting, when a connection goes bad. The default value is three connection retries.

-cs mlsrv12 option

Specifies connection parameters for your MobiLink System Database (MLSD).

Syntax

mlsrv12 -c "*connection-string***" -cs "***connection-string***"** …

Remarks

MobiLink server system objects, such as system tables, procedures, triggers, and views can be stored in a database other than the consolidated database. The database that stores the MobiLink system objects is called MLSD.

When this command option is specified on the command line, the MobiLink server makes connections to MLSD to fetch user defined scripts and to maintain synchronization status, such as ML user names, remote IDs, progress offsets, last upload and download timestamps, etc. The MobiLink server uses the original -c command line option connections to the consolidated database to upload data from and download data to the client databases. The consolidated database does not need to have any of the MobiLink server system objects. All the user defined scripts, including the error reporting and error handling scripts, are fetched from the MLSD and executed in the consolidated database.

When this option is used, the MobiLink server requires the Microsoft Distributed Transaction Coordinator (MSDTC).

The consolidated database and MLSD can be any one of the supported MobiLink consolidated databases. However, the corresponding ODBC drivers must support Microsoft Distributed Transactions.

The consolidated database and MLSD must have a transaction log to use MSDTC.

This option can only be used on Windows operating systems.

-ct mlsrv12 option

Sets the length of time, in minutes, that a connection may be unused before it is timed out and disconnected by the MobiLink server.

Syntax

mlsrv12 -c "*connection-string***" -ct** *connection-timeout* …

Remarks

MobiLink database connections that go unused for a specified amount of time are freed by the server. The timeout can be set using the -ct option. A default timeout period of 60 minutes is used.

-dl mlsrv12 option

Displays all MobiLink server messages on screen in the MobiLink server messages window.

Syntax

mlsrv12 -c "*connection-string***" -v -dl** …

Remarks

Display all MobiLink server messages in the MobiLink server messages window. By default, only a subset of all messages is shown in the window when a MobiLink server message log file is being output (using -o). In circumstances with many messages, this option can degrade performance.

See also

- "-o mlsrv12 option" on page 53
- "Log database server messages to a file" [*SQL Anywhere Server - Database Administration*]

-dr mlsrv12 option

For Adaptive Server Enterprise only. Ensures that tables involved in synchronization do not use the DataRow locking scheme.

Syntax

mlsrv12 -c "*connection-string***" -dr** …

Remarks

This option should only be used if none of the consolidated tables being synchronized were created using the DataRow locking scheme.

Use of this option reduces duplicate data sent by the MobiLink server.

See also

- "MobiLink isolation levels" on page 119

-ds mlsrv12 option

For use with restartable downloads. Specifies the maximum amount of data on disk that the MobiLink server can use to store all restartable downloads.

Syntax

mlsrv12 -c "*connection-string***" -ds** *size*[**k** | **m** | **g**] ...

Remarks

The MobiLink server holds download data that has not been received by the client for use in a restartable download. This option limits the amount of data that the server holds for all the synchronizations combined.

If *size* is too small the server may release download data, making it impossible to restart a download. The server does not release download data until one of the following occurs:

- The user successfully completes the download.
- The user comes back with a new synchronization request without resume enabled.
- The cache is needed for incoming requests. The oldest unsuccessful download is cleared first.

Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively. The default is **10m**.

See also

- "Resuming failed downloads" on page 114
- "-dc dbmlsync option" [*MobiLink - Client Administration*]
- "Resume Partial Download synchronization parameter" [*UltraLite - Database Management and Reference*]

-dsd mlsrv12 option

Disables snapshot isolation.

Syntax

mlsrv12 -c "*connection-string***" -dsd** ...

Remarks

When the consolidated database is SQL Anywhere (version 10 or later) or Microsoft SQL Server (2005 or later), the default isolation level for downloads is snapshot isolation. If the consolidated database is an earlier version of these databases, the default download isolation level is read committed.

You can also change the default isolation level in a script. However, for SQL Anywhere version 10 and Microsoft SQL Server 2005 and later databases, the isolation level is set at the start of the upload and download transactions. This means that if you set the isolation level in the begin_connection script, it may be overridden in the begin_upload and begin_download scripts.

This option only applies to SQL Anywhere version 10 and Microsoft SQL Server 2005 consolidated databases.

See also

- "MobiLink isolation levels" on page 119
- "-dt mlsrv12 option" on page 47
- "-esu mlsrv12 option" on page 48

-dt mlsrv12 option

For Microsoft SQL Server and Adaptive Server Enterprise databases only. Causes MobiLink to detect transactions only within the current database.

Syntax

mlsrv12 -c "*connection-string***" -dt** ...

Remarks

This option makes MobiLink ignore all transactions except ones within the current database. It increases throughput and reduces duplication of rows that are downloaded.

This option only affects timestamp-based downloads.

Use this option if:

- Your consolidated database is running on Microsoft SQL Server or Adaptive Server Enterprise that is also running other databases.
- You are using snapshot isolation for uploads or downloads with Microsoft SQL Server.
- You are using the DataRow locking scheme for synchronizing tables with Adaptive Server Enterprise.
- Your upload or download scripts do not access any other databases on the server.

This option only applies to Microsoft SQL Server databases using snapshot isolation, and Adaptive Server Enterprise databases using the DataRow locking scheme for tables involved in synchronization.

See also

- "MobiLink isolation levels" on page 119
- "-dsd mlsrv12 option" on page 46
- "-esu mlsrv12 option" on page 48

-e mlsrv12 option

Stores error logs sent from SQL Anywhere MobiLink clients.

Syntax

mlsrv12 -c "*connection-string***" -e** *filename* ...

Remarks

With no -e option, error logs from SQL Anywhere MobiLink clients are stored in a file named *mlsrv12.mle*. The -e option instructs the MobiLink server to store the error logs in the named file. By default, dbmlsync sends, on the occurrence of an error on the remote site, up to 32 kilobytes of remote log messages to a MobiLink server.

This option provides centralized access to remote error logs to help diagnose synchronization issues.

The amount of information delivered from a remote site can be controlled by the dbmlsync extended option ErrorLogSendLimit.

See also

- "-et mlsrv12 option" on page 49
- "ErrorLogSendLimit (el) extended option" [*MobiLink - Client Administration*]

-esu mlsrv12 option

Use snapshot isolation for uploads.

Syntax

mlsrv12 -c "*connection-string***" -esu** ...

Remarks

By default, MobiLink uses the read committed isolation level for uploads. This is usually the optimal isolation level.

If you use snapshot isolation for uploads, you may generate conflicts on snapshot transactions during upload updates. If this happens, the MobiLink server rolls back the entire upload and retries it. In this case, you might want to adjust your settings for the MobiLink server options -r or -rd to specify the delay time between retries and the maximum number of retries.

You can change the default isolation level in a script. To change the upload isolation level, you would typically use the begin_upload script.

This option only applies to SQL Anywhere version 10 and later and Microsoft SQL Server 2005 and later consolidated databases.

See also

- "MobiLink isolation levels" on page 119
- "-dsd mlsrv12 option" on page 46
- "-dt mlsrv12 option" on page 47
- "-r mlsrv12 option" on page 61
- "-rd mlsrv12 option" on page 62

-et mlsrv12 option

Stores error logs sent from SQL Anywhere MobiLink clients in the named file after truncating the existing file.

Syntax

mlsrv12 -c "*connection-string***" -et** *filename* …

Remarks

The -et option is the same as the -e option, except that the error log file is truncated before any new errors are added to it.

The amount of information delivered from a remote site can be controlled by the dbmlsync extended option ErrorLogSendLimit.

See also

- "ErrorLogSendLimit (el) extended option" [*MobiLink - Client Administration*]
- "-e mlsrv12 option" on page 48

-fips mlsrv12 option

Forces all secure MobiLink streams to be FIPS-compliant.

Syntax

mlsrv12 -c *connection-string***" -fips** …

Remarks

Specifying this option forces all MobiLink encryption to use FIPS-approved algorithms. You can still use unencrypted connections when the -fips option is specified, but you cannot use simple encryption.

When you use this option, FIPS-approved algorithms are used for connections regardless of whether you specify them or not. For example, if you start the MobiLink server with the option -fips and the option -x tls(...;fips=no;...), the fips=no setting is ignored and the server starts with fips=yes.

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*].

For MobiLink transport-layer security, the -fips option causes the server to use the FIPS-approved RSA encryption cipher, even if RSA without FIPS is specified. If ECC is specified, an error occurs because a FIPS-approved elliptic-curve algorithm is not available.

See also

- "Encrypting MobiLink client/server communications" [*SQL Anywhere Server - Database Administration*]
- "FIPS-approved encryption technology" [*SQL Anywhere Server - Database Administration*]

-ftr mlsrv12 option

Specifies a location for files that are to be downloaded by the mlfiletransfer utility or by the MobiLink Agent.

Syntax

mlsrv12 -c "*connection-string***" -ftr** *path* ...

Remarks

This option sets the file transfer root directory. Files that are to be transferred to a user can be placed in the root directory or in a subdirectory with the user name. MobiLink first looks for the requested file in a subdirectory of the file transfer root directory with the user name of the connected client. If the file is not in this subdirectory, MobiLink looks in the file transfer root directory.

This option is required if you want to use the mlfiletransfer utility to download files.

See also

- "MobiLink File Transfer utility (mlfiletransfer)" [*MobiLink - Client Administration*]
- "-ftru mlsrv12 option" on page 50
- "authenticate_file_transfer connection event" on page 309

-ftru mlsrv12 option

Specifies a location for files that are to be uploaded with the mlfiletransfer utility or by the MobiLink Agent.

Syntax

mlsrv12 -c "*connection-string***" -ftru** *path* ...

Remarks

This option sets the file transfer root directory for files to be uploaded with the mlfiletransfer utility. Files can only be uploaded into this root directory or immediate sub-directories of the root directory.

Files can only be uploaded if the authenticate_file_upload script does not exist or if the script exists and returns an authentication_code in the range 1000-1999. This requirement is for mlfiletransfer only and does not apply to the MobiLink Agent.

This option is required if you want to use the mlfiletransfer utility to upload files.

See also

- "MobiLink File Transfer utility (mlfiletransfer)" [*MobiLink - Client Administration*]
- "-ftr mlsrv12 option" on page 50
- "authenticate_file_transfer connection event" on page 309

-lsc mlsrv12 option

Specifies the connection information for the local server. This information is passed to other servers in the server farm.

Syntax

mlsrv12 -c "*connection-string***" -lsc** *protocol*[*protocol-options*] ...

protocol : **tcpip** | **tls** | **http** | **https**

protocol-options : **(** *option*=*value*; ... **)**

Remarks

This option is only used in the following situation:

- When running the notifier or QAnywhere in a MobiLink server farm.
- When using the mlreplay utility with the -rrp server option. See "-rrp mlsrv12 option" on page 62.

For example, if you have a server running on a host named server_rack10, the command line could start:

```
mlsrv12  -x tcpip(port=200) -zs server5 -lsc
tcpip(host=server_rack10;port=200) -c...
```

In this example, another server would use shared state in the consolidated database to get the connect string `tcpip(host=server_rack10;port=200)` and use it to connect to the server just started.

See also

- "Running the MobiLink server in a server farm" on page 29
- "-zs mlsrv12 option" on page 82
- "-rrp mlsrv12 option" on page 62
- "Notifiers in a MobiLink server farm" [*MobiLink - Server-Initiated Synchronization*]

-m mlsrv12 option

Enables QAnywhere messaging.

Syntax

mlsrv12 -c "*connection-string***" -m** [*message-properties-file*] ...

Remarks

The optional *message-properties-file* is deprecated. Properties are now specified via Sybase Central and stored in the consolidated database, or are specified with server management requests.

In the *message-properties-file*, each property must appear on its own line and consist of a property name, the = character, and then a property value.

For a list of properties you can set, see "Server properties" [*QAnywhere*].

See also

- "Introducing QAnywhere technology" [*QAnywhere*]
- "Starting QAnywhere with MobiLink enabled" [*QAnywhere*]
- "Server management requests" [*QAnywhere*]

Example

To start QAnywhere messaging when you are using the sample server message store (*samples-dir* *QAnywhere**server**qanyserv.db*), run the following command:

```
mlsrv12 -m -c "dsn=QAnywhere 10 Demo"
```

Note
For information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].

-nc mlsrv12 option

Sets the maximum number of concurrent network connections.

Syntax

mlsrv12 -c "*connection-string***" -nc** *connections* ...

Remarks

The MobiLink server rejects new synchronization connections when the limit is reached. On the client, a communication error is issued with a system error code that indicates the connection was refused.

The default is 1024.

To limit the number of concurrent synchronizations for non-persistent HTTP/HTTPS, set -nc significantly higher than -sm. When the -sm limit is reached, the MobiLink server provides an HTTP error 503

(Service Unavailable) to the remote client. If the -nc limit is reached, however, a socket error is issued. The greater the difference between -nc and -sm, the more likely it is that the rejected connections will generate the HTTP 503 error instead of the less descriptive socket error. For example, set -sm to 100 and set -nc to 1000. See "-sm mlsrv12 option" on page 67.

The maximum value for -nc depends on the operating system and its configuration. You may need to tune the configuration to achieve higher socket capacity.

-notifier mlsrv12 option

Starts the Notifier for server-initiated synchronization.

Syntax

mlsrv12 -c "*connection-string***" -notifier** [*notifier-properties-file*] ...

Remarks

If you specify a Notifier configuration file name, or if you do not specify a file name but you have a default Notifier properties file called *config.notifier*, the Notifier is configured using that file. This overrides any configuration information that is stored in the ml_properties table in the consolidated database.

Otherwise, MobiLink uses the configuration information that is stored in the ml_properties table in the consolidated database.

When you use the -notifier option, you start every Notifier that you have enabled.

For more information about enabling Notifiers, see "Notifier properties" [*MobiLink - Server-Initiated Synchronization*].

See also

- "MobiLink server settings for server-initiated synchronization" [*MobiLink - Server-Initiated Synchronization*]
- "Configuring server-side settings using the Notifier configuration file" [*MobiLink - Server-Initiated Synchronization*]
- "Notifiers" [*MobiLink - Server-Initiated Synchronization*]
- "Notifiers in a MobiLink server farm" [*MobiLink - Server-Initiated Synchronization*]

-o mlsrv12 option

Logs output messages to a MobiLink server message log file, and limits the data logged to the MobiLink server messages window.

Syntax

mlsrv12 -c "*connection-string***" -o** *logfile* ...

Remarks

Write all log messages to the specified file. Note that the MobiLink server messages window, if present, usually shows a subset of all messages logged.

The MobiLink server gives the full error context in its output file if errors occur during synchronization. The error context may include the following information:

- **Remote ID** This is the remote ID of the remote database synchronizing.
- **User Name** This is the actual user name that was provided to the MobiLink clients during synchronization.
- **Modified User Name** This is the user name as modified by the modify_user script.
- **Transaction** This lists the transaction the error occurs in. The transaction could be authenticate_user, begin_synchronization, upload, prepare_for_download, download, or end_synchronization.
- **Table Name** This shows the table name if it is available, or null.
- **Row Operation** The operation could be INSERT, UPDATE, DELETE or FETCH.
- **Row Data** This shows all the column values of the row that caused the error.
- **Script Version** This is the script version currently used for synchronization.
- **Script** This is the script that caused the error.

Error context information appears in the log regardless of your chosen level of verbosity.

See also

- "-os mlsrv12 option" on page 55
- "-dl mlsrv12 option" on page 45
- "-ot mlsrv12 option" on page 56
- "-on mlsrv12 option" on page 54
- "-v mlsrv12 option" on page 70

-on mlsrv12 option

Specifies a maximum size for the MobiLink server message log file, after which the file is renamed with the extension .old and a new file is started.

Syntax

mlsrv12 -c "*connection-string***" -on** *size* [**k** | **m**]...

Remarks

The *size* is the maximum file size for the message log, in bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively. The minimum size limit is 10 KB.

When the log file reaches the specified size, the MobiLink server renames the output file with the extension .old, and starts a new one with the original name.

Note
If the .old file already exists, it is overwritten. At most, two files will be used. To avoid losing old log files, use the -os option.

This option cannot be used with the -os option.

See also

- "-o mlsrv12 option" on page 53
- "-ot mlsrv12 option" on page 56
- "-on mlsrv12 option" on page 54
- "-os mlsrv12 option" on page 55
- "-v mlsrv12 option" on page 70

-oq mlsrv12 option

On Windows, prevents the appearance of the error window when a startup error occurs.

Syntax

mlsrv12 -c "*connection-string***" -oq** ...

Remarks

By default, the MobiLink server displays a window if a startup error occurs. The -oq option prevents this window from being displayed.

-os mlsrv12 option

Sets the maximum size of current and old MobiLink server message log files.

Syntax

mlsrv12 -c "*connection-string***" -os** *size* [**k** | **m**] ...

Remarks

The *size* is the maximum file size for logging output messages. The default unit is bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively. The minimum size limit is 10 KB.

Before the MobiLink server logs output messages to a file, it checks the current file size. If the log message makes the file size exceed the specified size, the MobiLink server renames the message log file to *yymmddxx.mls*, where *xx* is a number from 00 to 99, and *yymmdd* represents the current year, month, and day.

The latest output is always appended to the file specified by -o or -ot.

You cannot use this option with the -on option.

Note
This option makes an unlimited number of log files. To avoid this situation, use -o or -on.

See also

- "-o mlsrv12 option" on page 53
- "-on mlsrv12 option" on page 54
- "-ot mlsrv12 option" on page 56
- "-v mlsrv12 option" on page 70

-ot mlsrv12 option

Logs output messages to the MobiLink server message log file, but deletes the contents first.

Syntax

mlsrv12 -c "*connection-string***" -ot** *logfilename* ...

Remarks

The default is to send output to the MobiLink server message window or screen.

See also

- "-on mlsrv12 option" on page 54
- "-os mlsrv12 option" on page 55
- "-v mlsrv12 option" on page 70
- "-o mlsrv12 option" on page 53

-ppv mlsrv12 option

Causes MobiLink to print new periodic monitoring values according to the period specified. Periods are in seconds.

Syntax

mlsrv12 -c "*connection-string***" -ppv** *period* ...

Remarks

These values can provide insight into the state of the server, and are useful for determining the health and performance of the MobiLink server. For example, one could look at the DB_CONNECTIONS and LONGEST_DB_WAIT values to look for potential problems with the -w option or in the synchronization scripts. The values also provide an easy way to track system wide throughput measures, such as the number of rows uploaded or downloaded per second and the number of successful synchronizations per second.

The suggested period is 60 seconds.

If the period is set too small, the log will grow very quickly.

Each row of output is prefixed with **PERIODIC:** to aid in searching for and filtering out the values.

The printed values can include the following information:

Printed value	Description
CMD_PROCESSOR_STAGE_LEN	The length of the queue for synchronization work.
CPU_USAGE	The amount of CPU time used by the MobiLink server in microseconds.
DB_CONNECTIONS	The number of database connections in use.
FREE_DISK_SPACE	The disk space available on the temp disk in bytes.
HEARTBEAT_STAGE_LEN	The length of the queue for periodic, non-sync work.
LONGEST_DB_WAIT	The longest length of time an active synchronization has been waiting for the database.
LONGEST_SYNC	The age of the oldest synchronization in microseconds.
MEMORY_USED	The bytes of RAM in use (for Windows only).
ML_NUM_CONNECTED_CLIENTS	The number of connected synchronization clients.
NOTIFIER_STAGE_LEN	The length of the notifier work queue.
NUM_COMMITS	The total number of commits.
NUM_CONNECTED_FILE_XFERS	The number of mlfiletransfers currently connected.
NUM_CONNECTED_LISTENERS	The number of listeners currently connected.
NUM_CONNECTED_MONITORS	The number of monitors currently connected.
NUM_CONNECTED_PINGS	The number of pinging clients currently connected.
NUM_CONNECTED_SYNCS	The number of data synchronizations currently connected.
NUM_ERRORS	The total number of errors.
NUM_FAILED_SYNCS	The total number of failed syncs.
NUM_IN_APPLY_UPLOAD	The number of synchronizations currently in the apply upload phase.

Printed value	Description
NUM_IN_AUTH_USER	The number of synchronizations currently in the authenticate user phase.
NUM_IN_BEGIN_SYNC	The number of synchronizations currently in the begin synchronization phase.
NUM_IN_CONNECT	The number of synchronizations currently in the connect phase.
NUM_IN_CONNECT_FOR_ACK	The number of synchronizations currently in the connect for download ack phase.
NUM_IN_END_SYNC	The number of synchronizations currently in the end synchronization phase.
NUM_IN_FETCH_DNLD	The number of synchronizations currently in the fetch download phase.
NUM_IN_GET_DB_WORKER_FOR_ACK	The number of synchronizations currently waiting for a database connection to process a non-blocking download acknowledgement.
NUM_IN_NON_BLOCKING_ACK	The number of synchronizations currently in the non-blocking download ack phase.
NUM_IN_PREP_FOR_DNLD	The number of synchronizations currently in the prepare for download phase.
NUM_IN_RECVING_UPLOAD	The number of synchronizations currently in the receive upload phase.
NUM_IN_SEND_DNLD	The number of synchronizations currently in the send download phase.
NUM_IN_SYNC_REQUEST	The number of synchronizations currently in the synchronization request phase.
NUM_IN_WAIT_FOR_DNLD_ACK	The number of synchronizations currently in the wait for download ack phase.
NUM_ROLLBACKS	The total number of rollbacks.
NUM_ROWS_DOWNLOADED	The total number of rows sent to remotes.
NUM_ROWS_UPLOADED	The total number of rows received from remotes.
NUM_SUCCESS_SYNCS	The total number of successful syncs.

Printed value	Description
NUM_UNSUBMITTED_ER-ROR_RPTS	The number of unsubmitted error reports.
NUM_UPLOAD_CONNS_IN_USE	The number of upload connections currently in use.
NUM_WAITING_CONS	The number of synchronizations currently waiting for the consolidated database.
NUM_WARNINGS	The total number of warnings.
OE_STAGE_LEN	The length of the integrated Outbound Enabler work queue.
PAGES_IN_STREAMSTACK	The number of pages held by the network streams.
PAGES_LOCKED	The number of cache pages loaded into memory.
PAGES_LOCKED_MAX	The number of pages in the memory cache.
PAGES_SWAPPED_IN	The total number of pages ever read from disk.
PAGES_SWAPPED_OUT	The total number of pages ever swapped to disk.
PAGES_USED	The number of cache pages used. This includes pages swapped to disk so it may be larger than the cache size.
PRIMARY_IS_KNOWN	Indicates if the primary server is known or not. Shows 0 if the server does not care what the primary server is. Shows 1 if the server knows what the primary server. Shows 2 if the server does not know what the primary server is.
RAW_TCP_STAGE_LEN	The length of the network work queue.
SERVER_IS_PRIMARY	Indicates if the server is primary or secondary. Shows 1 if the server is primary, otherwise shows 0.
SIRT_NUM_LWP_HITS	The number of lightweight polls from remote task agents, indicating a notification.
SIRT_NUM_LWPS	The number of lightweight polls from remote task agents.
SIRT_NUM_REQUESTS	The number of remote task notifications currently outstanding.
STREAM_STAGE_LEN	The length of the high level network processing queue.
TCP_BYTES_READ	The total number of bytes ever read.

Printed value	Description
TCP_BYTES_WRITTEN	The total number of bytes ever written.
TCP_CONNECTIONS	The number of TCP connections currently opened.
TCP_CONNECTIONS_CLOSED	The total number of connections ever closed.
TCP_CONNECTIONS_OPENED	The total number of connections ever opened.
TCP_CONNECTIONS_REJECTED	The total number of connections ever rejected.
TIMED_WORK_STAGE_LEN	The length of the dynamic caching work queue.
TRACKED_MEMORY	The amount of memory allocated by the server. Use this metric for non-Windows systems where the MEMORY_USED metric is unavailable. On Microsoft Windows systems, use the MEMORY_USED metric for increased accuracy.
VM_MEM_USE	The amount of memory used by any attached VMs.

Example

Below is sample output showing the periodic monitoring values.

```
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_USED: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_LOCKED: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_LOCKED_MAX: 13243
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS_OPENED: 2
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS_CLOSED: 2
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS_REJECTED: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_BYTES_READ: 5137
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_BYTES_WRITTEN: 4549
I. 2009-10-28 11:46:29. <Main> PERIODIC: ML_NUM_CONNECTED_CLIENTS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_SWAPPED_OUT: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_SWAPPED_IN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_IN_STREAMSTACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: CPU_USAGE: 3359375
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_COMMITS: 7
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ROLLBACKS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_SUCCESS_SYNCS: 1
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_FAILED_SYNCS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ERRORS: 2
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_WARNINGS: 3
I. 2009-10-28 11:46:29. <Main> PERIODIC: DB_CONNECTIONS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: RAW_TCP_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: STREAM_STAGE_LEN: 5
I. 2009-10-28 11:46:29. <Main> PERIODIC: HEARTBEAT_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: CMD_PROCESSOR_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ROWS_DOWNLOADED: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ROWS_UPLOADED: 7
I. 2009-10-28 11:46:29. <Main> PERIODIC: FREE_DISK_SPACE: 124154904576
I. 2009-10-28 11:46:29. <Main> PERIODIC: LONGEST_DB_WAIT: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: LONGEST_SYNC: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_UNSUBMITTED_ERROR_RPTS: 193
I. 2009-10-28 11:46:29. <Main> PERIODIC: MEMORY_USED: 140275712
```



```
I. 2009-10-28 11:46:29. <Main> PERIODIC: SERVER_IS_PRIMARY: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_SYNCS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_PINGS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_FILE_XFERS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_MONITORS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_LISTENERS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_WAITING_CONS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_SYNC_REQUEST: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_RECVING_UPLOAD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_CONNECT: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_AUTH_USER: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_BEGIN_SYNC: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_APPLY_UPLOAD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_PREP_FOR_DNLD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_FETCH_DNLD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_END_SYNC: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_SEND_DNLD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_WAIT_FOR_DNLD_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_GET_DB_WORKER_FOR_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_CONNECT_FOR_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_NON_BLOCKING_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_UPLOAD_CONNS_IN_USE: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: TRACKED_MEMORY: 56269577
I. 2009-10-28 11:46:29. <Main> PERIODIC: VM_MEM_USE: 517013504
I. 2009-10-28 11:46:29. <Main> PERIODIC: TIMED_WORK_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NOTIFIER_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: SIRT_NUM_REQUESTS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: SIRT_NUM_LWPS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: SIRT_NUM_LWP_HITS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: OE_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PRIMARY_IS_KNOWN: 0
```

-q mlsrv12 option

Instructs MobiLink to run with a minimized messages window on startup.

Syntax

mlsrv12 -c "*connection-string***" -q** ...

Remarks

Minimize the MobiLink server messages window.

-r mlsrv12 option

Sets the maximum number of deadlock retries.

Syntax

mlsrv12 -c "*connection-string***" -r** *retries* ...

Remarks

By default, MobiLink server retries uploads that are deadlocked in the consolidated database for a maximum of 10 attempts. If the deadlock is not broken, synchronization fails, since there is no guarantee

that the deadlock can be overcome. This option allows an arbitrary retry limit to be set. To stop the server from retrying deadlocked transactions, specify **-r 0**. The upper bound on this setting is 2 to the power 32, minus one.

Note
Deadlocks should not be part of a normal synchronizations system. If encountered, they should be eliminated by fixing your synchronizations scripts.

-rd mlsrv12 option

Sets the maximum delay time between deadlock retries.

Syntax

mlsrv12 -c "*connection-string***" -rd** *delay* ...

Remarks

When upload transactions are deadlocked in the consolidated database, the MobiLink server waits a random length of time before retrying the transaction. The random nature of the delay increases the likelihood that future attempts succeed. This option allows you to specify the maximum delay in units of seconds. The value 0 (zero) makes retries instantaneous, but larger values are recommended because they yield more successful retries. The default and maximum delay value is **30**.

Note
Deadlocks should not be part of a normal synchronizations system. If encountered, they should be eliminated by fixing your synchronizations scripts.

-rrp mlsrv12 option

Causes the MobiLink server to run the mlreplay utility and replay all recorded sessions (files with extension *mlr*) in the given directory when the server starts.

Use this option to preload remote schemas into the MobiLink server. This saves the time and effort for the first synchronizing remotes in the field to send the remote schema.

Syntax

mlsrv12 -c "*connection-string***" -rrp** *directory* ...

Remarks

To use the -rrp option, a local server connection string must be specified using the -lsc option, so the mlreplay utility can connect to the server. See “-lsc mlsrv12 option” on page 51.

Follow the procedure below to use -rrp:

To use the -rp and -rrp options

1. Record synchronizations using the -rp option.
2. Determine which prerecorded synchronizations to use to preload schema. There should be one for each schema and/or set of publications.
3. Copy the prerecorded synchronizations to a new directory.
4. Run in production without the -rp option and with the -rrp option.

See also

- "-rp mlsrv12 option" on page 63
- "-lsc mlsrv12 option" on page 51
- "MobiLink replay utility (mlreplay)" on page 733

-rp mlsrv12 option

Specifies the directory to which synchronizations are recorded for playback with the mlreplay utility.

Syntax

mlsrv12 -c "*connection-string***" -rp** *directory* …

Remarks

For the best performance, use this option to record synchronizations used by the -rrp option. The -rrp option enables all synchronizations, including the first synchronization of each unique schema, to take advantage of the schema cache. See "-rrp mlsrv12 option" on page 62.

Follow the procedure below to use -rp:

To use the -rp and -rrp options

1. Record synchronizations using the -rp option.
2. Determine which prerecorded synchronizations to use to preload schema. There should be one for each schema and/or set of publications.
3. Copy the prerecorded synchronizations to a new directory.
4. Run in production without the -rp option and with the -rrp option.

See also

- "-rrp mlsrv12 option" on page 62
- "MobiLink replay utility (mlreplay)" on page 733

-s mlsrv12 option

Sets the maximum number of rows that can be uploaded at the same time.

Syntax

mlsrv12 -c "*connection-string***" -s** *count* ...

Remarks

Set the maximum number of rows that can be inserted, updated, or deleted at the same time to *count*.

The MobiLink server sends upload rows to the consolidated database through the ODBC driver. This option controls the number of rows sent to the database server in each batch. Increasing this value can speed up processing of the upload stream and reduce network time. However, with a higher setting the MobiLink server may require more resources for applying the upload stream.

The number of rows uploaded at once can be viewed in the log file as **rowset size**.

The default is 10.

-sl dnet mlsrv12 option

Sets the .NET Common Language Runtime (CLR) options and forces the CLR to load on startup. This option is recommended when using .NET scripting logic.

Syntax

mlsrv12 -c "*connection-string***" -sl dnet** *options* ...

Remarks

Sets options to pass directly to the .NET CLR. The options are:

Option	Description
-D*name=value*	Set an environment variable. For example, `-Dsynchtype=far -Dextra_rows=yes` For more information, see the .NET framework class System.Environment.
-MLAutoLoadPath= *path*	Set the location of base assemblies. Only works with private assemblies. To tell MobiLink where assemblies are located, use this option or -MLDomConfigFile, but not both. When you use -MLAutoLoadPath, you cannot specify a domain in the event script. The default is the current directory.

Option	Description
-MLDomConfigFile= *file*	Set the location of base assemblies. Use when you have shared assemblies, or you don't want to load all assemblies in the directory, or you cannot use MLAutoLoadPath for some other reason. To tell MobiLink where assemblies are located, use -MLDomConfigFile or -MLAutoLoadPath, but not both.
-MLStartClasses= *classnames*	At server startup, load and instantiate user-defined start classes in the order listed.
-clrConGC	Enable concurrent garbage collection in the CLR.
-clrFlavor=(**wks** | **svr**)	Flavor of the .NET CLR to load. The flavor is **svr** for server and **wks** for workstation. By default, **svr** is loaded.
-clrVersion= *version*	Version of the .NET CLR to load. This must be prefixed with **v**. For example, **v1.0.3705** loads the directory *\Microsoft.NET \Framework\v1.0.3705*.

To display this list of options, run the following command:

```
mlsrv12 -sl dnet (?)
```

See also

- "Writing synchronization scripts in .NET" on page 588

-sl java mlsrv12 option

Sets the Java VM options and forces the Java VM to load on startup. This option is recommended when using Java scripting logic. On Unix, the -cp options must be separated with colons.

Syntax

mlsrv12 -c "*connection-string***" -sl java (** *options* **)** ...

Remarks

Sets -jrepath and other options to pass directly to the Java VM. The options are:

Option	Description
-classic	Use the classic Java VM.
-client	Use the client Java VM.
-hotspot	Use the hotspot Java VM.

Option	Description
-server	Use the server Java VM. This is the default.
-cp *location*;...	Specify a set of directories or JAR files in which to search for classes. You can also use -classpath.
-D*name=value*	Set a system property. For example, `-Dsynchtype=far -Dextra_rows=yes`
-DMLStartClasses=*classname*, ...	At server startup, load and instantiate user-defined start classes in the order listed.
-jrepath *path*	Override the default JRE path, which is the directory *install-dir\Sun\jre160_chip* (where chip is a supported chip, such as x86).
-verbose [**:class** |**:gc** | **:jni**]	Enable verbose output.
-X *vm-option*	Set a VM-specific option as described in the file *install-dir\Sun\jre160_chip\bin\client\Xusage.txt* (where chip is a supported chip, such as x86).

To display a list of Java options you can use, type:

```
java
```

Unix notes

Options must be enclosed in brackets. These can be round brackets, as shown in the syntax above, or curly braces { }.

The -jrepath option is only available on Windows. On Unix, if you want to load a specific JRE, you should set the LD_LIBRARY_PATH (LIBPATH on IBM AIX, SHLIB_PATH on HP-UX) to include the directory containing the JRE. The directory must be listed before any of the SQL Anywhere installation directories.

See also

- "Writing synchronization scripts in Java" on page 523

Examples

For example, on Windows the following partial mlsrv12 command line sets the Java VM option that enables system asserts:

```
mlsrv12 -sl java (-cp ;\myclasses; -esa) ...
```

On Windows, the following partial mlsrv12 command line defines the LDAP_SERVER system property:

```
mlsrv12 -sl java ( -cp ;\myclasses; -DLDAP_SERVER=mycorp-ldap ) ...
```


The following partial mlsrv12 command line works on Unix:

```
mlsrv12 -sl java { -cp .:$CLASSPATH:/opt/myclasses:/opt/my.jar: }
```

-sm mlsrv12 option

Sets the maximum number of synchronizations that can be actively worked on by limiting the maximum number of network connections.

Syntax

mlsrv12 -c "*connection-string***" -sm** *number* ...

Remarks

The MobiLink server performs the following synchronization tasks simultaneously:

1. Read upload data from the network and unpack it.
2. Apply uploads to the consolidated database.
3. Fetch rows to be downloaded from the consolidated database.
4. Pack download data and send it to remote databases.

The number of synchronizations for each task is limited as follows:

- The number of synchronizations doing tasks 2 and 3 is less than or equal to the setting for the mlsrv12 -w option.
- The number of synchronizations doing task 2 is less than or equal to the setting for the mlsrv12 -wu option.
- The number of synchronizations doing all four tasks is less than or equal to the setting for the -sm option.

Higher values for -sm, especially when much greater than -w, allow the MobiLink server to perform more network tasks (1 and 4) than database tasks (2 and 3). This can help ensure that a database worker doesn't have to wait for tasks when network performance might otherwise be a bottleneck. This can improve throughput. However, if -sm is set too high and there are enough concurrent connections, the MobiLink server can allocate more memory than is directly available, causing the virtual memory paging of the operating system to be activated, which in turn causes memory to be swapped to disk—significantly decreasing throughput.

See also

- “-w mlsrv12 option” on page 74
- “-wu mlsrv12 option” on page 75
- “-nc mlsrv12 option” on page 52

-tc mlsrv12 option

Sets a timeout threshold for long running SQL scripts.

Syntax

mlsrv12 -c "*connection string***" -tc** *minutes* ...

Remarks

By default, the MobiLink server watches the execution time of each SQL script and issues a warning message when the execution time of the script reaches 10 minutes. Long running scripts are more likely to cause contention and blocking in the consolidated database, which can significantly reduce overall throughput.

You can use the -tf option to cancel statements that exceed the threshold.

The default value can be reset to zero or a positive integer and its units are in minutes. When it is set to zero, the -tc option is disabled and the MobiLink server does not watch any script execution.

When the timeout threshold is a non-zero value, the MobiLink server shows the warning message in an exponential way. The warning is shown when the execution time first passes the time specified; the warning is shown again when the execution time passes 2 times the given time, then 4 times the given time, and so on.

The warning message contains the connection ID used for the current synchronization and a context that includes the following, if they are available: Remote ID, ML User Name, Modified User Name, Transaction, Table Name, Row Values and Script Version. The timeout warning context is shown regardless of the verbose settings of the MobiLink server.

When the consolidated database is running on an Oracle database server and the timeout warning message occurs, a database user with DBA authority may need to check the consolidated database to determine the cause of the problem. The ServiceName and SERIAL# of the connection used by the synchronization can be found in the warning message. If the synchronization connection is stopped, the MobiLink server terminates the current synchronization.

See also

- "-tf mlsrv12 option" on page 68

-tf mlsrv12 option

This option is used to let the MobiLink server fail a SQL script if the execution time passes the timeout specified by -tc. This option is not available when the consolidated database is running on an Oracle server.

Syntax

mlsrv12 -c "*connection string***" -tf** ...

Remarks

If the SQL script fails, the MobiLink server will either skip the row (if the script is an upload script and if the handle_error script returns 1000) and continue the synchronization, or abort the synchronization.

The MobiLink server shows a warning message if this option is specified and it is running against an Oracle server.

This option is ignored if **-tc 0** is specified.

-tx mlsrv12 option

When using transactional uploads, this option batches groups of transactions and commits them together.

Syntax

mlsrv12 -c "*connection-string***" -tx** *count* ...

Remarks

Use this option to improve performance when doing transactional uploads.

count can be any non-negative value. The default is 1, which means commit every transaction separately. Use a value of zero to perform one commit after all transactions have been uploaded.

The ideal value for *count* can only be determined through performance testing.

See also

- "-tu dbmlsync option" [*MobiLink - Client Administration*]

-ud mlsrv12 option

Instructs MobiLink to run as a daemon.

Syntax

mlsrv12 -c "*connection-string***" -ud** ...

Remarks

This option applies to Unix platforms only.

See also

- "Running the MobiLink server outside the current session" on page 24

-ui mlsrv12 option

For Linux with X window server support, starts the MobiLink server in shell mode if a usable display isn't available.

Syntax

mlsrv12 -c "*connection-string***" -ui** ...

Remarks

When -ui is specified, the server attempts to find a usable display. If it cannot find one, for example because the X window server isn't running, then the MobiLink server starts in shell mode.

-ux mlsrv12 option

For Linux, opens the MobiLink server messages window where messages are displayed.

Syntax

mlsrv12 -c "*connection-string* **-ux** ...

Remarks

When -ux is specified, the MobiLink server must be able to find a usable display. If it cannot find one, for example because the DISPLAY environment variable is not set or because the X window server is not running, the MobiLink server fails to start.

To run the MobiLink server messages window in quiet mode, use -q.

On Windows, the MobiLink server messages window appears automatically.

See also

- "-q mlsrv12 option" on page 61

-v mlsrv12 option

Allows you to specify what information is logged to the message log file.

Syntax

mlsrv12 -c "*connection-string***" -v**[*levels*] ...

Remarks

This option controls the type of messages written to the message log file.

If you specify -v alone, the MobiLink server writes a minimal amount of information about each synchronization. The more levels specified, the more verbose the output to the message log file.

A high level of verbosity can adversely affect performance and should only be used during development.

The MobiLink server can be set to use different log verbosity for a targeted MobiLink user or remote ID. The MobiLink server checks the ml_property table every five minutes and looks for verbose settings for a MobiLink user or remote ID. See "Log verbosity for targeted MobiLink users and remote IDs" on page 706.

When a CHAR, VARCHAR, NCHAR or NVARCHAR column with a byte length of greater than 32767 bytes is synchronized, the MobiLink server does not display the full contents of the column values in verbosity. Instead, the first chunk of data, up to 100 bytes in length, is displayed. This applies to the i, q and r levels.

The available levels are as follows. You can use one or more of these options at once; for example, -vnrsu.

- **+** Turn on all of the lower case verbosity levels.
- **c** Show the content of each synchronization script when it is invoked. This level implies s.
- **e** Show system event scripts. These system event scripts are used to query and maintain MobiLink system tables.
- **f** Show first-read errors. This logs errors caused when load-balancing devices check for server liveness by making connections that don't send any data, and cause failed synchronizations. Use this option to verify that the load balancer is properly performing liveness checks.

  See also the TCP/IP option **ignore**. For more information, see "-x mlsrv12 option" on page 75.
- **h** Show the remote schema being synchronized.
- **i** Display the column values of each row uploaded. Use this option instead of -vr, which displays the column values of each row uploaded and downloaded, to reduce the amount of data being logged. Specifying -vi with -vq is the same as specifying -vr.
- **k** Prints a line to the log whenever the cache grows or shrinks. This can be used during load testing to find the optimal cache size which can then be used as a static cache size when deploying your system.
- **m** Prints the duration of each synchronization and the duration of each synchronization phase to the log whenever a synchronization completes. The synchronization phases are shown below. They are the same as those displayed in the MobiLink Monitor. All times are shown in milliseconds (ms).
  - **Synchronization request** The time taken between creating the network connection between the MobiLink client and the MobiLink server, up to receiving the first bytes of the upload stream.
  - **Receive upload** The time taken from the first bytes of the upload stream being received by the MobiLink server until the upload stream from the MobiLink client has been completely received. The time may be significant even for a download-only synchronization. The time depends on the size of the upload stream and the network bandwidth for the transfer.
  - **Get DB worker** The time taken to acquire a free database worker thread.

- **Connect** The time taken by the database worker thread to make a database connection if a new database connection is needed. For example, after an error on the previous connection or if the script version has changed.
- **Authenticate user** The time taken to authenticate the user.
- **Begin synchronization** The time taken for the begin_synchronization event if it is defined, plus the time to fetch the last_upload_time for each subscription.
- **Apply upload** The time taken for the uploaded data to be applied to the consolidated database.
- **Prepare for download** The time taken for the prepare_for_download event.
- **Fetch download** The time taken to fetch the rows to be downloaded from the consolidated database to create the download stream.
- **End synchronization** The time taken for the end_synchronization event, after which the database worker thread is released. This phase occurs before the download stream is sent to the remote database.
- **Send download** The time taken to send the download stream to the remote database. The time depends on the size of the download stream and the network bandwidth for the transfer. For an upload-only synchronization, the download stream is simply an upload acknowledgement.
- **Wait for download ack** The time spent waiting for the download to be applied to the remote database and for the remote database to send the download acknowledgement. This phase is only shown if the MobiLink client has enabled download acknowledgement.
- **Get DB worker for download ack** The time spent waiting for a free database worker thread after the download acknowledgement has been received. This phase is only shown if the MobiLink client has enabled download acknowledgement.
- **Connect for download ack** The time required by the database worker thread to make a database connection if a new database connection is needed. This phase is only shown if the MobiLink client has enabled download acknowledgement.
- **Non-blocking download ack** The time required for the publication_nonblocking_download_ack connection and nonblocking_download_ack connection events. This phase is only shown if the MobiLink client has enabled download acknowledgement.

Each value is prefixed with "PHASE:" to aid in searching for the values.

The following example is sample output showing the durations for the various synchronization phases:

```
I. 2008-06-05 14:48:36. <1> PHASE: start_time: 2008-06-05 14:48:36.048
I. 2008-06-05 14:48:36. <1> PHASE: duration: 175
I. 2008-06-05 14:48:36. <1> PHASE: sync_request: 0
I. 2008-06-05 14:48:36. <1> PHASE: receive_upload: 19
I. 2008-06-05 14:48:36. <1> PHASE: get_db_worker: 0
I. 2008-06-05 14:48:36. <1> PHASE: connect: 18
I. 2008-06-05 14:48:36. <1> PHASE: authenticate_user: 51
I. 2008-06-05 14:48:36. <1> PHASE: begin_sync: 69
I. 2008-06-05 14:48:36. <1> PHASE: apply_upload: 0
```



```
I. 2008-06-05 14:48:36. <1> PHASE: prepare_for_download: 1
I. 2008-06-05 14:48:36. <1> PHASE: fetch_download: 4
I. 2008-06-05 14:48:36. <1> PHASE: wait_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: end_sync: 0
I. 2008-06-05 14:48:36. <1> PHASE: send_download: 10
I. 2008-06-05 14:48:36. <1> PHASE: get_db_worker_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: connect_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: nonblocking_download_ack: 0
```

- **n** Show row-count totals per synchronization.
- **o** Show SQL passthrough activity.
- **p** Show both remote and consolidated progress offsets per synchronization.
- **q** Display the column values of each row downloaded. Use this option instead of -vr, which displays the column values of each row uploaded and downloaded, to reduce the amount of data being logged. Specifying -vi with -vq is the same as specifying -vr.
- **r** Display the column values of each row uploaded or downloaded. To log only the column values of each row uploaded, use -vi. To log only the column values of each row downloaded, use -vq.
- **R** For synchronizations only, show the remote ID in each log message. The MobiLink server adds the prefix `yyyy-mm-dd hh:mm:ss. <sync_id> (remote_id,)` to the log entries.

  Use this option with the -vU option to also show the user name in the log message.

  These two command line options are not affected by the -v+ option, that is, the MobiLink server does not add the remote ID or the MobiLink user name into its logging messages even if the -v+ option is used.
- **s** Show the name of each synchronization script as it is invoked.
- **t** Show the translated SQL that results from scripts that are written in ODBC canonical format. This level implies c. The following example shows the automatic translation of a statement for SQL Anywhere.

  ```
  I. 2009-02-11 11:02:14. [102]: begin_upload synch2
  { call SynchLogLine( ?, ?, 'begin_upload' ) }
  I. 2009-02-11 11:02:14. [102]: Translated SQL:
  call SynchLogLine( ?, ?, 'begin_upload' )
  ```

  The following example shows the translation of the same statement for Microsoft SQL Server.

  ```
  I. 2009-02-11 11:03:21. [102]: begin_upload synch2
  { call SynchLogLine( ?, ?, 'begin_upload' ) }
  I. 2009-02-11 11:03:21. [102]: Translated SQL:
  EXEC SynchLogLine ?, ?, 'begin_upload'
  ```

- **u** Show undefined table scripts. This may help new users understand the synchronization process and the flow of events.
- **U** For synchronizations only, shows the user name in each log message. The MobiLink server adds the prefix `yyyy-mm-dd hh:mm:ss. <sync_id> (,user_name)` to the log entries.

Use this option with the -vR option to also show the remote ID in the log message.

These two command line options are not affected by the -v+ option, that is, the MobiLink server does not add the remote ID or the MobiLink user name into its logging messages even if the -v+ option is used.

See also

- "MobiLink statistical properties" on page 168

-w mlsrv12 option

Sets the number of database worker threads.

Syntax

mlsrv12 -c "*connection-string***" -w** *count* ...

Remarks

Each database worker thread accepts synchronization requests one at a time, but also concurrently with all other database worker threads.

Each database worker thread uses one connection to the consolidated database. The MobiLink server opens one additional connection for administrative purposes. So, the minimum number of connections from the MobiLink server to the consolidated database is *count* + 1.

The number of database worker threads has a strong influence on MobiLink synchronization throughput, and you need to run tests to determine the optimum number for your particular synchronization setup. The number of database worker threads determines how many synchronizations can be active in the consolidated database simultaneously; the rest gets queued waiting for database worker threads to become available. Adding database worker threads *may* increase throughput, but it also increases the possibility of contention between the active synchronizations. At some point adding more database worker threads decreases throughput because the increased contention outweighs the benefit of overlapping synchronizations.

The value set for this option is also the default setting for the -wu option, which can be used to limit the number of threads that can simultaneously upload to the consolidated database. This is useful if the optimum number of database worker threads for downloading is larger than the optimum number for uploading. The best throughput may be achieved with a large number of database worker threads (via -w) with a small number allowed to apply uploads simultaneously (via -wu). In general, the optimum number for -wu depends on the consolidated database, and is relatively independent of the processing or network speeds for the remote databases. Therefore, when you increase the number of threads with -w, you may want to use -wu to restrict the number that can upload simultaneously. For more information, see "-wu mlsrv12 option" on page 75.

The default number of database worker threads is **5**.

See also

- "-wu mlsrv12 option" on page 75
- "-sm mlsrv12 option" on page 67
- "-cn mlsrv12 option" on page 43

-wu mlsrv12 option

Sets the maximum number of database worker threads that can apply uploads to the consolidated database simultaneously.

Syntax

mlsrv12 -c "*connection-string***" -wu** *count* ...

Remarks

Use the -wu option to limit the number of database worker threads that can simultaneously apply uploads to the consolidated database. When the limit is reached, a database worker thread that is ready to apply its upload to the consolidated database must wait until another finishes its upload.

The most common cause of contention in the consolidated database is having too many database worker threads applying uploads simultaneously. Downloads usually cause far less contention, so they are limited only by the mlsrv12 -w option. For this reason, the -w setting must be greater than or equal to the -wu setting.

By default, all database worker threads can apply uploads simultaneously. The number of database worker threads that are used is set by the -w option. The default is **5**.

Example

In a pilot setup using a LAN and remote databases on PCs, you find that the optimum number of database worker threads is approximately 10 for both upload-only and download-only synchronizations, and that corresponds to 100% CPU utilization on the consolidated database. With fewer database worker threads you find that throughput is less and the CPU utilization for the consolidated database is lower. With more database worker threads, throughput does not increase because the consolidated database is already processing as fast as it can with 10 workers.

See also

- "-w mlsrv12 option" on page 74
- "-sm mlsrv12 option" on page 67

-x mlsrv12 option

Sets network protocol options used by the MobiLink server to listen for synchronization requests.

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*].

Syntax 1

mlsrv12 -c "*connection-string***" -x** *protocol*[**(***protocol-options***)**] ...

protocol : **tcpip** | **tls** | **http** | **https** | **oe**

protocol-options : **(** *option***=***value*; ... **)**

Default

The default is TCPIP with port 2439.

Parameters

The allowed values of protocol are as follows:

- **tcpip** Accept connections using TCP/IP.
- **tls** Accept connections using TCP/IP and transport-layer security (TLS).
- **http** Accept connections using the standard HTTP Web protocol.
- **https** Accept connections using a variant of HTTP that handles secure transactions. The HTTPS protocol implements HTTP over SSL/TLS using RSA or ECC encryption.
- **oe** Use an integrated Outbound Enabler when using the Relay Server. You cannot include more than one **-x oe** switch on the server command line.

You can also specify the following network protocol options, in the form *option*=*value*. You must separate multiple options with semicolons.

- **TCP/IP options** If you specify the tcpip protocol, you can optionally specify the following protocol options (these options are case sensitive):

TCP/IP protocol option	Description
host=*hostname*	The host name or IP number on which the MobiLink server should listen. The default value is localhost.

TCP/IP protocol option	Description
ignore=*hostname*	A host name or IP number that gets ignored by the MobiLink server if it makes a connection. This option allows you to ignore requests from load balancers at the lowest possible level, preventing excessive output in the MobiLink server log and MobiLink Monitor output files. You can specify multiple hosts to ignore; for example `-x tcpip(ignore=lb1;ignore=123.45.67.89)`. If you specify multiple instances of -x on a command line, the host is ignored on all instances; for example, if you specify -x tcpip(ignore=1.1.1.1) -x http, then connections for 1.1.1.1 are ignored on both the TCP/IP and the HTTP streams.
port=*portnumber*	The socket port number on which the MobiLink server should listen. The default port is 2439, which is the IANA registered port number for the MobiLink server.

- **Options for TCP/IP with transport-layer security** If you specify the tls protocol, which is TCP/IP with transport-layer security, you can optionally specify the following protocol options (these options are case sensitive):

TLS protocol options	Description
fips={**yes**|**no**}	If you specify the TLS protocol with tls_type=rsa, you can specify fips=yes to accept connections using the TCP/IP protocol and FIPS-approved algorithms for encryption. FIPS connections use separate FIPS 140-2 certified software. Servers using RSA encryption without FIPS are compatible with clients using RSA with FIPS, and servers using RSA with FIPS are compatible with clients using RSA without FIPS.
host=*hostname*	The host name or IP number on which the MobiLink server should listen. The default value is localhost.
ignore=*hostname*	A host name or IP number that gets ignored by the MobiLink server if it makes a connection. This option allows you to ignore requests from load balancers at the lowest possible level, preventing excessive output in the MobiLink server log and MobiLink Monitor output files. You can specify multiple hosts to ignore; for example `-x tcpip(ignore=lb1;ignore=123.45.67.89)`.
port=*portnumber*	The socket port number on which the MobiLink server should listen. The default port is 2439, which is the IANA registered port number for the MobiLink server.

TLS protocol options	Description
tls_type={**rsa**|**ecc**}	If you specify the protocol as tls, you can specify either elliptic-curve cryptography (ecc) or RSA encryption (rsa). For backward compatibility, ecc can also be specified as certicom. The default tls_type is rsa. When you use TLS, you must specify an identity and an identity password: ○ **identity=*identity-file*** Specify the path and file name of the identity file that is to be used for server authentication. ○ **identity_password=*password*** Specify the password for the identity. See "Starting the MobiLink server with transport-layer security" [*SQL Anywhere Server - Database Administration*].
e2ee_type={**rsa**|**ecc**}	The type of the key used to exchange session keys for end-to-end encryption. Must be either rsa or ecc, and must match the key type in the private key file (see next option). The default e2ee_type is rsa.
e2ee_private_key=*file*	The PEM-encoded file containing the rsa or ecc private key. This option is required for end-to-end encryption to take effect. PEM-encoded files are created using the createkey utility. See "Key Pair Generator utility (createkey)" [*SQL Anywhere Server - Database Administration*].
e2ee_private_key_password=*password*	The password to the private key file. This option is required for end-to-end encryption to take effect. To avoid making this password visible in the MobiLink server command line, use the dbfhide utility. See "File Hiding utility (dbfhide)" [*SQL Anywhere Server - Database Administration*].

- **HTTP options** If you specify the http protocol, you can optionally specify the following protocol options (these options are case sensitive):

HTTP options	Description
buffer_size=*number*	The maximum body size for an HTTP message sent from MobiLink server, in bytes. Changing the option decreases or increases the amount of memory allocated for sending HTTP messages. The default is 65536 bytes.
host=*hostname*	The host name or IP number on which the MobiLink server should listen. The default value is localhost.

HTTP options	Description
log_bad_request={**yes**|**no**}	When set to yes, the MobiLink server prints an error if it receives an incomplete or unexpected HTTP request. These errors are analogous to those printed by the -vf option. The default is no. See "-v mlsrv12 option" on page 70.
port=*portnumber*	The socket port number on which the MobiLink server should listen. The default port is 80.
version=*http-version*	The MobiLink server automatically detects the HTTP version used by a client. This parameter is a string specifying the default version of HTTP to use if the server cannot detect the version used by the client. You have a choice of 1.0 or 1.1. The default value is 1.1.

- **HTTPS options** The HTTPS protocol uses RSA or ECC digital certificates for transport-layer security. If you specify FIPS encryption, the protocol uses separate FIPS 140-2 certified software that is compatible with HTTPS.

  For more information, see "Starting the MobiLink server with transport-layer security" [*SQL Anywhere Server - Database Administration*].

  If you specify the **https** protocol, you can optionally specify the following protocol options (these options are case sensitive):

HTTPS options	Description
buffer_size=*number*	The maximum body size for an HTTPS message sent from MobiLink server, in bytes. Changing the option decreases or increases the amount of memory allocated for sending HTTPS messages. The default is 65536 bytes.
identity=*server-identity*	The path and file name of the identity file that is to be used for server authentication.
identity_password=*password*	An optional parameter that specifies a password for the identity file. See "Transport-layer security" [*SQL Anywhere Server - Database Administration*]. To avoid making this password visible in the MobiLink server command line, use the dbfhide utility. See "File Hiding utility (dbfhide)" [*SQL Anywhere Server - Database Administration*].
fips={**yes**|**no**}	You can specify fips=yes to accept connections using the HTTPS protocol and FIPS-approved algorithms for encryption. FIPS connections use separate FIPS 140-2 certified software. Servers using RSA encryption without FIPS are compatible with clients using RSA with FIPS, and servers using RSA with FIPS are compatible with clients using RSA without FIPS.

HTTPS options	Description
host= *hostname*	The host name or IP number on which the MobiLink server should listen. The default value is localhost.
port=*portnumber*	The socket port number on which the MobiLink server should listen. The port number must match the port the MobiLink server is set up to monitor. The default port is 443.
tls_type={**rsa**|**ecc**}	If you specify the TCP/IP protocol as tls, you can specify either elliptic-curve cryptography (ecc) or RSA encryption (rsa). For backward compatibility, ecc can also be specified as certicom. The default tls_type is rsa. When you use transport-layer security, you must specify an identity and an identity password: ○ **identity=*identity-file*** Specify the path and file name of the identity file that is to be used for server authentication. ○ **identity_password=password** Specify the password for the identity file. See "Starting the MobiLink server with transport-layer security" [*SQL Anywhere Server - Database Administration*].
version=*http-version*	The MobiLink server automatically detects the HTTP version used by a client. This parameter is a string specifying the default version of HTTP to use if the server cannot detect the version used by the client. You have a choice of 1.0 or 1.1. The default value is 1.1.
e2ee_type={**rsa**|**ecc**}	The type of the key used to exchange session keys for end-to-end encryption. Must be either rsa or ecc, and must match the key type in the private key file (see next option). The default e2ee_type is rsa.
e2ee_private_key=*file*	The PEM-encoded file containing the rsa or ecc private key. This option is required for end-to-end encryption to take effect. PEM-encoded files are created using the createkey utility. See "Key Pair Generator utility (createkey)" [*SQL Anywhere Server - Database Administration*].
e2ee_private_key_password=*password*	The password to the private key file. This option is required for end-to-end encryption to take effect. To avoid making this password visible in the MobiLink server command line, use the dbfhide utility. See "File Hiding utility (dbfhide)" [*SQL Anywhere Server - Database Administration*].

HTTPS options	Description
log_bad_request={yes|no}	When set to yes, the MobiLink server prints an error if it receives an incomplete or unexpected HTTP request. These errors are analogous to those printed by the -vf option. The default is no. See "-v mlsrv12 option" on page 70.

- **OE options** If you specify the oe protocol, you can optionally specify the following protocol options (these options are case sensitive):

OE protocol option	Description
config=*file*	The Outbound Enabler configuration file. It has the same format as a configuration file for the stand-alone Outbound Enabler, except you should not specify the -cs switch within the file. You must specify exactly one configuration file.
buffer_size=*number*	The maximum body size for an HTTP message sent from MobiLink server, in bytes. Changing the option decreases or increases the amount of memory allocated for sending HTTP messages. The default is 65536 bytes.
version=*http-version*	The MobiLink server automatically detects the HTTP version used by a client. This parameter is a string specifying the default version of HTTP to use if the server cannot detect the method used by the client. You have a choice of 1.0 or 1.1. The default value is 1.1.
log_bad_request={yes|no}	When set to yes, the MobiLink server prints an error if it receives an incomplete or unexpected HTTP request. These errors are analogous to those printed by the -vf option. The default is no. See "-v mlsrv12 option" on page 70.

Example

The following command line sets the TCP/IP port to 12345:

```
mlsrv12 -c "dsn=SQL Anywhere 12 CustDB;uid=DBA;pwd=sql" -x tcpip(port=12345)
```

The following example specifies the type of security (RSA), the server identity file, and the identity password protecting the server's private key:

```
mlsrv12 -c "dsn=my_cons"
 -x tls(tls_type=rsa;identity=c:\test\serv_rsa1.crt;identity_password=pwd)
```

The following example is similar to the previous, except that there is a space in the identity file name:

```
mlsrv12 -c "dsn=my_cons"
 -x "tls(tls_type=rsa;identity=c:\Program Files\test
\serv_rsa1.crt;identity_password=pwd)"
```

The following example shows the use of end-to-end encryption over HTTPS:

```
mlsrv12 -c "dsn=my_cons" -x https(tls_type=rsa;identity=my_identity.crt;
identity_password=my_id_pwd;e2ee_type=rsa;e2ee_private_key=my_pk.pem;
e2ee_private_key_password=my_pk_pwd)
```

The following example shows the use of the integrated Outbound Enabler:

```
mlsrv12 -c "dsn=my_cons" -x oe(config=oeconfig.txt)
```

In the example above, the *oeconfig.txt* file contains the following:

```
-f farm.name
-id servername
-t a18b2e37dbc296322ec5a6eded6ba896
-cr "host=relayserver.sybase.com;https=0;port=80"
```

-zf mlsrv12 option

Causes the MobiLink server to check for script changes at the beginning of each synchronization.

Syntax

mlsrv12 -c "*connection-string***" -zf**

Remarks

Unless the -zf option is used, the MobiLink server assumes that no script changes have been made and does not check for script changes after it is started.

-zp mlsrv12 option

Adjusts the precision of timestamp comparisons for the purpose of conflict detection.

Syntax

mlsrv12 -c "*connection-string***" -zp**

Remarks

This option causes MobiLink server to use the highest timestamp resolution representable in both remote and consolidated databases when comparing timestamps for conflict detection purposes. The option is useful when timestamps in the consolidated database are more precise than in the remote, as updated timestamps on the remote can cause spurious conflicts in the next synchronization. This option allows MobiLink to ignore these conflicts. When there is a precision mismatch and -zp is not used, a per synchronization and a schema sensitive per table warning are written to the log to advertise the -zp option. Another per synchronization warning is also added to tell users to adjust the timestamp precision on the remote database where possible.

-zs mlsrv12 option

Specifies a MobiLink server name for mlstop.

Syntax

mlsrv12 -c "*connection-string***" -zs** *name*

Remarks

The default name is `<default>`.

The name that is specified may include ASCII letters and numbers, but no other characters.

When mlstop is used to shut down a MobiLink server started with the -zs option, you must specify the server name on the mlstop command line. For example, `mlstop myMLserver`. Shutdown may only be initiated from the computer where the MobiLink server is installed.

See also

- "MobiLink stop utility (mlstop)" on page 731

-zt mlsrv12 option

Specifies the maximum number of processors used to run the MobiLink server.

Syntax

mlsrv12 -c "*connection-string***" -zt** *number*

Remarks

This option may be required for some ODBC drivers. It also gives you fine control of processor resources.

This option can only be used on Windows and Linux operating systems. The default is the number of processors on the computer.

-zu mlsrv12 option

Controls the automatic addition of users when the authenticate_user and authenticate_user_hashed scripts are undefined.

Syntax

mlsrv12 -c "*connection-string***" -zu**{ + | - } ...

Remarks

If this is supplied as -zu+, then unrecognized MobiLink user names are added automatically to the ml_user table on first synchronizing. If the argument is supplied as -zu-, or not supplied, unrecognized user names are prevented from synchronizing.

This option is useful during development to register users. It is not recommended for deployed applications.

See also

- “Synchronizations from new users” [*MobiLink - Client Administration*]
- “MobiLink users” [*MobiLink - Client Administration*]
- “MobiLink user authentication utility (mluser)” on page 732
- “authenticate_user connection event” on page 315

-zus mlsrv12 option

Causes the MobiLink server to invoke upload scripts for a table even when no rows are uploaded for the table.

Syntax

mlsrv12 -c "*connection-string***" -zus** …

Remarks

By default, if no rows are uploaded for a table, the MobiLink server does not invoke upload scripts for that table, even if they are defined. This option overrides the default behavior and causes the MobiLink server to call upload scripts for a table even if no rows are uploaded.

-zw mlsrv12 option

Controls which levels of warning message to display.

Syntax

mlsrv12 -c "*connection-string***" -zw** *levels*

Remarks

MobiLink has five levels of warning messages:

Level	Description
0	Suppress all warning messages
1	Server and high ODBC level: warning messages when the MobiLink server starts
2	Synchronization and user level: warning messages when a synchronization starts
3	Schema level: warning messages when a MobiLink server is processing a client schema

Level	Description
4	Script and lower ODBC level: warning messages when a MobiLink server fetches, prepares, or executes scripts
5	Table or row level: warning messages when a MobiLink server performs table operations in an upload or download

To specify the level of warning messages you want reported, you can separate levels with a comma, or separate a range with two dots. For example, **-zw 1..3,5** is the same as **-zw 1,2,3,5**.

The reporting of messages has a slight impact on performance. Levels with a higher number tend to produce more messages.

If -zw is used more than once in the same command line, MobiLink recognizes only the last instance. If settings of -zw, -zwd, and -zwe conflict, MobiLink gives priority to -zwe, then -zwd, then -zw.

The default is **1,2,3,4,5**, which indicates that all levels of warning message should be displayed.

-zwd mlsrv12 option

Disables specific warning codes.

Syntax

mlsrv12 -c "*connection-string***" -zwd** *code*, ...

Remarks

You can disable specific warning codes so that they do not get reported, even though other codes of the same level are reported.

For a complete list of warning message codes, see "MobiLink server warning messages" [*Error Messages*].

If -zwd is used more than once in the same command line, MobiLink accumulates the settings. If settings of -zw, -zwd, and -zwe conflict, MobiLink gives priority to -zwe, then -zwd, then -zw.

-zwe mlsrv12 option

Enables specific warning codes.

Syntax

mlsrv12 -c "*connection-string***" -zwe** *code*, ...

Remarks

You can enable specific warning codes so that they are reported even though you have disabled other codes of the same level using -zw.

For a complete list of warning message codes, see "MobiLink server warning messages" [*Error Messages*].

If -zwe is used more than once on the same command line, MobiLink accumulates the settings. If settings of -zw, -zwd, and -zwe conflict, MobiLink gives priority to -zwe, then -zwd, then -zw.

Synchronization techniques

MobiLink development tips

Adding synchronization functionality to an application adds a degree of complexity to your application. While the added complexity is almost always manageable, you need to be aware of it. The entire synchronization system, from the remotes through to the consolidated database, including other consolidated database applications, has many parts and each requires attention. The following tips may be useful.

When you are adding synchronization to a prototype application, it can be difficult to see which components are causing problems, so start with a prototype without synchronization. Once your prototype is working correctly, only then do you enable synchronization.

Start with straightforward synchronization techniques. Operations such as a simple upload or download require only one or two scripts. Once those are working correctly, you can introduce more advanced techniques, such as timestamps, primary key pools, conflict resolution, and arbitrary business logic.

MobiLink and primary keys

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, MobiLink applications 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 in synchronized tables.
- Primary keys in synchronized tables must be unique across all synchronized databases.

  See "Maintaining unique primary keys" on page 96.

Timestamp-based downloads

The timestamp method is the most useful general technique for efficient downloads. This technique involves tracking the last time that each user synchronized and only downloading rows that have changed since then.

MobiLink maintains a timestamp value indicating when each MobiLink user last downloaded data. This value is called the **last download time**.

See "Using last download times in scripts" on page 88.

To implement timestamp-based synchronization for a table

1. At the consolidated database, add a last_modified column that holds the most recent time the row was modified. The column is typically declared as follows:

DBMS	last modified column
Adaptive Server Enterprise	`datetime`
IBM DB2 LUW	`timestamp`
Microsoft SQL Server	`datetime`
MySQL	`timestamp`
Oracle	`timestamp`
SQL Anywhere	`timestamp DEFAULT timestamp`

2. In scripts for the download_cursor and download_delete_cursor events, compare the first parameter to the value in the timestamp column.

Example

The following table declaration and scripts implement timestamp-based synchronization on the Customer table in the Contact sample:

- Table definition:

```
CREATE TABLE "DBA"."Customer"(
   "cust_id"  integer NOT NULL DEFAULT GLOBAL AUTOINCREMENT,
   "name"  char(40) NOT NULL,
   "rep_id"  integer NOT NULL,
   "last_modified" timestamp NULL DEFAULT timestamp,
   "active"  bit NOT NULL,
   PRIMARY KEY ("cust_id") )
```

- download_cursor script:

```
SELECT cust_id, Customer.name, Customer.rep_id
FROM Customer KEY JOIN SalesRep
WHERE Customer.last_modified >= {ml s.last_table_download}
    AND SalesRep.ml_username = {ml s.username}
    AND Customer.active = 1
```

See "Synchronization logic source code" [*MobiLink - Getting Started*] and "Synchronizing contacts in the Contact sample" [*MobiLink - Getting Started*].

Using last download times in scripts

The last download timestamp is provided as a parameter to many MobiLink events. The last download timestamp is the value obtained from the consolidated database during the last successful synchronization immediately before the download phase. If the current MobiLink user has never synchronized, or has never synchronized successfully, this value is set to 1900-01-01.

See "How download timestamps are generated and used" on page 88.

If you have multiple publications and have synchronized them at different times, then you can have several different last download timestamps. For this reason, there are two script parameter names for last download timestamps:

- **last_table_download** is the last download timestamp for the current table being synchronized.
- **last_download** is the last time all tables were synchronized. It is the earliest last_table_download value for any table being synchronized.

When you use question marks instead of named parameters in MobiLink scripts, the correct value is always used, based on the event. Using question marks in SQL scripts has been deprecated and it is recommended that you use named parameters.

Caution
The column holding the last modified information should not be synchronized. If your remote databases require such a column, a different column name should be used. Otherwise, the timestamp value may be overridden by the uploaded value, and would not contain the time that the row was last modified in the consolidated database.

See also

- "Script parameters" on page 268

Example

```
SELECT cust_id, Customer.name, Customer.rep_id
FROM Customer KEY JOIN SalesRep
WHERE Customer.last_modified >= {ml s.last_table_download}
    AND SalesRep.ml_username = {ml s.username}
    AND Customer.active = 1
```

How download timestamps are generated and used

MobiLink generates and uses a timestamp for timestamp-based downloads as follows:

- After an upload is committed and immediately before invoking the prepare_for_download event, the MobiLink server fetches the current time from the consolidated database and saves the value. This timestamp value represents the start time of the current download; the next synchronization should only download data that changes after this time.

  Note
  If the consolidated database supports snapshot isolation, then the download timestamp is the minimum of:

  - the current time
  - the start of the oldest open transaction

- The MobiLink server sends this timestamp value as part of the download, and the client stores it.
- The next time the client synchronizes, it uses the timestamp value for the **last_download_timestamp** that it sends with the upload.
- The MobiLink server passes the last_download_timestamp that the client just uploaded into your download scripts. Your scripts can then select changes with timestamps that are newer or equal to the last last_download_timestamp to ensure that only new changes are downloaded.

Where the last download time is stored

The last download time is stored on the remote database. This is the appropriate place because only the remote knows if the download has been successfully applied.

For SQL Anywhere remotes, the last download time is stored per subscription. See "SYSSYNC system view" [*SQL Anywhere Server - SQL Reference*].

For UltraLite remotes, the last download time is stored per publication. See "syspublication system table" [*UltraLite - Database Management and Reference*].

Changing the last download time

In some rare circumstances you may want to modify the last_download_timestamp. For example, if you accidentally delete all the data on a remote database, you can download it again by defining a modify_last_download_timestamp connection script to reset the value for the last download timestamp. There are other events, called generate_next_last_download_timestamp and modify_next_last_download_timestamp, which you can use to set the timestamp not for the current synchronization but for the next synchronization. For example, if you wanted to use a UTC timestamp value to compare to UTC values in the last_modified columns of your tables. See:

- "modify_last_download_timestamp connection event" on page 402
- "generate_next_last_download_timestamp event" on page 381
- "modify_next_last_download_timestamp connection event" on page 405

UltraLite also provides functionality to change the last download time from the remote. See:

- C/C++ embedded SQL: "ULResetLastDownloadTime method" [*UltraLite - C and C++ Programming*]
- C++: "ResetLastDownloadTime method" [*UltraLite - C and C++ Programming*]
- .NET: "ResetLastDownloadTime method" [*UltraLite - .NET Programming*]

See also

- "Using last download times in scripts" on page 88

Dealing with daylight savings time

Daylight savings time can cause problems in a distributed database system if data is synchronized during the hour that the time changes. In fact, you can lose data. This is only an issue in the autumn when the time goes back and there is a one-hour period that can be ambiguous.

To deal with daylight savings time, you have four possible solutions:

- Ensure that the consolidated database server is using UTC time.
- Turn off daylight savings time on the consolidated database server.
- Shut down for an hour when the time changes.
- Use UTC timestamps in your download timestamp columns and use either a generate_next_last_download_timestamp or modify_next_last_download_timestamp script to provide a UTC timestamp for the next last download timestamp.

Snapshot synchronization

Snapshot synchronization of a table is a complete download of all relevant rows in the table, even if they have been downloaded before. This is the simplest synchronization method, but can involve unnecessarily large data sets being exchanged, which can limit performance and could also cost more in telecom charges.

You can use snapshot synchronization for downloading all the rows of the table, or in conjunction with a partitioning of the rows. See "Partitioning rows among remote databases" on page 92.

When to use snapshot synchronization

The snapshot method is typically most useful for tables that have both the following characteristics.

- **Relatively few rows** When there are few rows, the overhead for downloading all rows is small.
- **Rows that change frequently** When most rows in a table change frequently, there is little to be gained by explicitly excluding those that have not changed since the last synchronization.

A table that holds a list of exchange rates could be suited to this approach because there are relatively few currencies, but the rates of most change frequently. Depending on the nature of the business, a table that holds prices, a list of interest rates, or current news items could all be candidates.

To implement snapshot-based synchronization

1. Leave the upload scripts undefined unless remote users update the values.

2. If the table may have rows deleted, write a download_delete_cursor script that deletes all the rows from the remote table, or at least all rows no longer required. For the latter approach, do not delete the rows from the consolidated database; rather, mark them for deletion. You must know the row values to delete them from the remote database.

   See "Writing download_delete_cursor scripts" on page 293.

3. Write a download_cursor script that selects all the rows you want to include in the remote table.

Deleting rows when using snapshot synchronization

Rather than deleting rows from the consolidated database, mark them for deletion. You must know the row values to delete them from the remote database. Select only unmarked rows in the download_cursor script and only marked rows in the download_delete_cursor script.

The download_delete_cursor script is executed before the download_cursor script. If a row is to be included in the download, you need not include a row with the same primary key in the delete list. When a downloaded row is received at the remote location, it replaces a preexisting row with the same primary key.

See "Writing scripts to download rows" on page 290.

An alternative deletion technique

Rather than delete rows from the remote database using a download_cursor script, you can allow the remote application to delete the rows. For example, immediately following synchronization, you could allow the application to execute SQL statements that delete the unnecessary rows.

Rows deleted by the application are ordinarily uploaded to the MobiLink server upon the next synchronization, but you can prevent this upload using the STOP SYNCHRONIZATION DELETE statement. For example,

```
STOP SYNCHRONIZATION DELETE;
DELETE FROM table-name
 WHERE expiry_date < CURRENT TIMESTAMP;
COMMIT;
START SYNCHRONIZATION DELETE;
```

See "Writing download_delete_cursor scripts" on page 293.

Snapshot example

The ULProduct table in the sample application is maintained by snapshot synchronization. The table contains relatively few rows, and for this reason, there is little overhead in using snapshot synchronization.

1. There is no upload script. This reflects a business decision that products cannot be added at remote databases.

2. There is no download_delete_cursor, reflecting an assumption that products are not removed from the list.

3. The download_cursor script selects the product identifier, price, and name of every current product. If the product is pre-existing, the price in the remote table is updated. If the product is new, a row is inserted in the remote table.

   ```
   SELECT prod_id, price, prod_name
   FROM ULProduct
   ```

For another example of snapshot synchronization in a table with very few rows, see "Synchronizing sales representatives in the Contact sample" [*MobiLink - Getting Started*].

Partitioning rows among remote databases

Each MobiLink remote database can contain a different subset of the data in the consolidated database. This means that you can write your synchronization scripts so that data is **partitioned** among remote databases.

The partitioning can be disjoint, or it can contain overlaps. For example, if each employee has their own set of customers, with no shared customers, the partitioning is **disjoint**. If there are shared customers who appear in more than one remote database, the partitioning contains **overlaps**.

Partitioning is implemented in the download_cursor and download_delete_cursor scripts for the table, which define the rows to be downloaded to the remote database. Each of these scripts takes a MobiLink user name as a parameter. By defining your scripts using this parameter in the WHERE clause, each user gets the appropriate rows.

Disjoint partitioning

Partitioning is controlled by the download_cursor and download_delete_cursor scripts for each table involved in synchronization. These scripts make sure of two parameters, a last download timestamp and the MobiLink user name supplied in the call to synchronize.

To partition a table among remote databases

1. Include in the table definition a column containing the synchronization user name in the consolidated database. You need not download this column to remote databases.

2. Include a condition in the WHERE clause of the download_cursor and download_delete_cursor scripts requiring this column to match the script parameter.

   The script parameter is represented by a named parameter in the script. For example, the following download_cursor script partitions the Contact table by employee ID.


```
SELECT id, contact_name
FROM Contact
WHERE last_modified >= {ml s.last_table_download}
AND emp_id = {ml s.username}
```

See "download_cursor table event" on page 347 and "download_delete_cursor table event" on page 349.

Example

The primary key pool tables in the CustDB sample application are used to supply each remote database with its own set of primary key values. This technique is used to avoid duplicate primary keys, and is discussed in "Using primary key pools" on page 100.

A necessary feature of the method is that primary key-pool tables must be partitioned among remote databases in a disjoint fashion.

One key-pool table is ULCustomerIDPool, which holds primary key values for each user to use when they add customers. The table has three columns:

- **pool_cust_id** A primary key value for use in the ULCustomer table. This is the only column downloaded to the remote database.
- **pool_emp_id** The employee who owns this primary key.
- **last_modified** This table is maintained using the timestamp technique, based on the last_modified column.

  For information about timestamp synchronization, see "Timestamp-based downloads" on page 86.

The download_cursor script for this table is as follows.

```
SELECT pool_cust_id
FROM ULCustomerIDPool
WHERE last_modified >= {ml s.last_table_download}
 AND pool_emp_id = {ml s.username}
```

See "Synchronizing customers in the Contact sample" [*MobiLink - Getting Started*] and "Synchronizing contacts in the Contact sample" [*MobiLink - Getting Started*].

Partitioning with overlaps

Some tables in your consolidated database may have rows that belong to many remote databases. Each remote database has a subset of the rows in the consolidated database and the subset overlaps with other remote databases. This is frequently the case with a customer table. In this case, there is a many-to-many relationship between the table and the remote databases and there is usually a table to represent the relationship. The scripts for the download_cursor and download_delete_cursor events need to join the table being downloaded to the relationship table.

Example

The CustDB sample application uses this technique for the ULOrder table. The ULEmpCust table holds the many-to-many relationship information between ULCustomer and ULEmployee.

Each remote database receives only those rows from the ULOrder table for which the value of the emp_id column matches the MobiLink user name.

The SQL Anywhere version of the download_cursor script for ULOrder in the CustDB application is as follows:

```
SELECT o.order_id, o.cust_id, o.prod_id,
   o.emp_id, o.disc, o.quant, o.notes, o.status
FROM ULOrder o , ULEmpCust ec
WHERE o.cust_id = ec.cust_id
 AND ec.emp_id = {ml s.username}
 AND ( o.last_modified >= {ml s.last_table_download}
  OR ec.last_modified >= {ml s.last_table_download})
 AND  ( o.status IS NULL
  OR o.status != 'Approved' )
 AND ( ec.action IS NULL )
```

This script is fairly complex. It illustrates that the query defining a table in the remote database can include more than one table in the consolidated database. The script downloads all rows in ULOrder for which the following are all true:

- the cust_id column in ULOrder matches the cust_id column in ULEmpCust
- the emp_id column in ULEmpCust matches the synchronization user name
- the last modification of either the order or the employee-customer relationship was later than the most recent synchronization time for this user
- the status is anything other than **Approved**

The action column on ULEmpCust is used to mark columns for delete. When NULL, the row is deemed to be fully active (not deleted).

The download_delete_cursor script is as follows.

```
SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc, o.quant, o.notes,
o.status
   FROM ULOrder o, dba.ULEmpCust ec
   WHERE o.cust_id = ec.cust_id
     AND ( ( o.status = ''Approved'' AND o.last_modified >= {ml
s.last_table_download} )
        OR ( ec.action = ''D''  )  )
     AND ec.emp_id = {ml s.username}
```

This script deletes all approved rows from the remote database.

Partitioning child tables

The example in the previous section illustrates how to partition tables based on a criterion in some other table. See "Partitioning with overlaps" on page 93.

Some tables in your remote database may have disjoint subsets or overlapping subsets, but do not contain a column that determines the subset. These are child tables that usually have a foreign key (or a series of foreign keys) referencing another table. The referenced table has a column that determines the correct subset.

In this case, the download_cursor script and the download_delete_cursor script need to join the referenced tables and have a WHERE clause that restricts the rows to the correct subset.

For an example, see the Customer table's download scripts "Synchronizing contacts in the Contact sample" [*MobiLink - Getting Started*].

Upload-only and download-only synchronizations

By default, synchronization is bi-directional: data is both uploaded and downloaded. However, you can choose to do only an upload or only a download.

Synchronization model note
This topic provides information for how to set up upload-only and download-only synchronization when you create your MobiLink synchronization system in your database. You can also specify upload-only or download-only if you create a synchronization model in Sybase Central.

SQL Anywhere remote databases

- **Upload** To perform upload-only synchronization, use the dbmlsync option -uo or the extended option UploadOnly. See:
  - "-uo dbmlsync option" [*MobiLink - Client Administration*]
  - "UploadOnly (uo) extended option" [*MobiLink - Client Administration*]
- **Download** To perform download-only synchronization, use the dbmlsync option -ds or the extended option DownloadOnly. See:
  - "-ds dbmlsync option" [*MobiLink - Client Administration*]
  - "DownloadOnly (ds) extended option" [*MobiLink - Client Administration*]

  SQL Anywhere remote databases can also use download-only publications. This approach to downloads is different from download-only synchronizations. See "Download-only publications" [*MobiLink - Client Administration*].

UltraLite remote databases

- **Upload** To perform upload-only synchronization, use the Upload Only synchronization parameter.

  See "Upload Only synchronization parameter" [*UltraLite - Database Management and Reference*].
- **Download** To perform download-only synchronization, use the Download Only synchronization parameter.

See "Download Only synchronization parameter" [*UltraLite - Database Management and Reference*].

Maintaining unique primary keys

Every table that is to be synchronized must have a primary key, and for each synchronized table the primary key must be unique across all synchronized databases. The values of primary keys should not be updated.

It is often convenient to use a single column as the primary key for tables. For example, each customer should be assigned a unique identification value. If all the sales representatives work in an environment where they can maintain a direct connection to the database, assigning these numbers is easily accomplished. Whenever a new customer is inserted into the customer table, automatically add a new primary key value that is greater than the last value.

In a disconnected environment, assigning unique values for primary keys when new rows are inserted is not as easy. When a sales representative adds a new customer, she is doing so to a remote copy of the Customer table. You must prevent other sales representatives, working on other copies of the Customer table, from using the same customer identification value.

This section describes the following ways to solve the problem of how to generate unique primary keys across all synchronized databases:

- "Using composite keys" on page 96
- "Using UUIDs" on page 96
- "Using global autoincrement" on page 97
- "Using primary key pools" on page 100

Using composite keys

The MobiLink remote ID uniquely defines a remote database within a synchronization system. Therefore, an easy way to create a unique primary key is to create a composite primary key that includes the MobiLink remote ID as part of its value. If you maintain unique MobiLink user names, you could use the user name instead of the remote ID.

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

Using UUIDs

You can ensure that primary keys are unique by using the newid( ) function to create universally unique values for your primary key. The resulting UUIDs can be converted to a string using the uuidtostr( ) function, and converted back to binary using the strtouuid( ) function.

UUIDs, also known as GUIDs, are unique across all computers. However, the values are completely random and so cannot be used to determine when a value was added, or the order of values. UUID values

are also considerably larger than the values required by other methods (including global autoincrement), and require more table space in both the primary and foreign key tables. Indexes on tables using UUIDs are also less efficient.

See also

SQL Anywhere databases:

- "The NEWID default" [*SQL Anywhere Server - SQL Usage*]
- "NEWID function [Miscellaneous]" [*SQL Anywhere Server - SQL Reference*]
- "UNIQUEIDENTIFIER data type" [*SQL Anywhere Server - SQL Reference*]

UltraLite databases:

- "Primary key uniqueness in UltraLite" [*UltraLite - Database Management and Reference*]
- "NEWID function [Miscellaneous]" [*UltraLite - Database Management and Reference*]

Example

The following SQL Anywhere CREATE TABLE statement creates a primary key that is universally unique:

```
CREATE TABLE customer (
   cust_key UNIQUEIDENTIFIER NOT NULL
            DEFAULT NEWID( ),
   rep_key VARCHAR(5),
   PRIMARY KEY(cust_key))
```

Using global autoincrement

In SQL Anywhere and UltraLite databases, you can set the default column value to be GLOBAL AUTOINCREMENT. You can use this default for any column in which you want to maintain unique values, but it is particularly useful for primary keys.

Global autoincrement values are partitioned among remote databases in contiguous ranges of values. The set of possible values is finite, so the larger the size of the each range, the fewer ranges are available. Care must be taken to set the correct size of the range for your needs. Exhausting a range is possible, but you can detect this and assign a new range. See "How default values are chosen" on page 99.

To use global autoincrement columns

1. Declare the column as a global autoincrement column.

   When you specify default global autoincrement, the domain of values for that column is partitioned. Each partition contains the same number of values. For example, if you set the partition size for an integer column in a database to 1000, one partition extends from 1001 to 2000, the next from 2001 to 3000, and so on.

   See "Declaring default global autoincrement" on page 98.

2. Set the global_database_id value.

SQL Anywhere and UltraLite databases supply default values in a database only from the partition uniquely identified by that database's number. For example, if you assign a database the identity number 10 and the partition size is 1000, the default values in that database would be chosen in the range 10001-11000. Another copy of the database, assigned the identification number 11, would supply default value for the same column in the range 11001-12000.

See "Setting the global database ID" on page 98.

Declaring default global autoincrement

You can set default values in your database by selecting the column properties in Sybase Central, or by including the DEFAULT GLOBAL AUTOINCREMENT phrase in a CREATE TABLE or ALTER TABLE statement.

Optionally, the partition size can be specified in parentheses immediately following the AUTOINCREMENT keyword. The partition size may be any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition is rarely, if ever, 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, especially if your column is not of type INT or BIGINT, it is best to specify the partition size explicitly.

For example, the following SQL statement creates a simple table with two columns: an integer that holds a customer identification number and a character string that holds the customer's name. The partition size is set to 5000, which would be appropriate for an application database where few new rows are inserted in each remote database.

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

See also

- SQL Anywhere: "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- UltraLite: "CREATE TABLE statement [UltraLite] [UltraLiteJ]" [*UltraLite - Database Management and Reference*]

Setting the global database ID

When deploying an application, you must assign a different identification number to each database. You can create and distribute the identification numbers by a variety of means. One method is to place the values in a table and download the correct row to each database based on some other unique property, such as remote ID.

To set the global database identification number

- In SQL Anywhere, you set the global ID of a database by setting the value of the public option global_database_id. The identification number must be a non-negative integer. See "global_database_id option" [*SQL Anywhere Server - Database Administration*].

  In UltraLite, you set the global ID of a database by setting the global_id option. See "UltraLite global_database_id option" [*UltraLite - Database Management and Reference*].

How default values are chosen

The global database ID is set with the public option global_database_id in SQL Anywhere, and with the global_id option in UltraLite.

The global database id option in each database must be set to a unique, non-negative integer. The range of default values for a particular database is $pn + 1$ to $p(n + 1)$, where p is the partition size and n is the value of the global database ID. For example, if the partition size is 1000 and global database ID is set to 3, then the range is from 3001 to 4000.

SQL Anywhere and UltraLite choose default values by applying the following rules:

- If the column contains no values in the current partition, the first default value is $pn + 1$, where p is the partition size and n is the value of the global database ID.
- If the column contains values in the current partition, but all are less than $p(n + 1)$, the next default value is 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.

If the global database ID is set to the default value of 2147483647, a null value is inserted into the column. Should null values not be permitted, the attempt to insert the row causes an error. This situation arises, for example, if the column is contained in the table's primary key.

Because the global database ID cannot be set to negative values, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size.

Null default values are also generated when the supply of values within the partition has been exhausted. In this case, a new unique global database ID value should be assigned to the database to allow default values to be chosen from another partition. Attempting to insert the null value causes an error if the column does not permit nulls. To detect that the supply of unused values is low and handle this condition in SQL Anywhere databases, you can create an event of type GlobalAutoincrement.

Should the values in a particular partition become exhausted, you can assign a new global database ID to that database. You can assign new database ID numbers in any convenient manner. However, one possible technique is to maintain a pool of unused database ID values. This pool is maintained in the same manner as a pool of primary keys. See "Using primary key pools" on page 100.

You can set an event handler to automatically notify the database administrator (or perform some other action) when the partition is nearly exhausted. For SQL Anywhere databases, see "Defining trigger conditions for events" [*SQL Anywhere Server - Database Administration*].

See also

- "Setting the global database ID" on page 98
- SQL Anywhere: "global_database_id option" [*SQL Anywhere Server - Database Administration*]
- UltraLite: "UltraLite global_database_id option" [*UltraLite - Database Management and Reference*]

Example

In a SQL Anywhere database, the following statement sets the database identification number to 20.

```
SET OPTION PUBLIC.global_database_id = 20
```

If the partition size for a particular column is 5000, default values for this database are selected from the range 100001-105000.

Using primary key pools

One efficient means of solving the problem of unique primary keys is to assign each user of the database a pool of primary key values that can be used as the need arises. For example, you can assign each sales representative 100 new identification values. Each sales representative can freely assign values to new customers from his or her own pool.

To implement a primary key pool

1. Add a new table to the consolidated database and to each remote database to hold the new primary key pool. Apart from a column for the unique value in the consolidated database, these tables should contain a column for a user name, to identify who has been given the right to assign the value.

2. In the consolidated database, write a stored procedure to ensure that each user is assigned enough new identification values. Assign more new values to remote users who insert many new entries or who synchronize infrequently.

3. Write a download_cursor script to select the new values assigned to each user and download them to the remote database. See "Timestamp-based downloads" on page 86.

4. Modify the application that uses the remote database so that when a user inserts a new row, the application uses one of the values from the pool. The application must then delete that value from the pool so it is not used a second time.

5. Write an upload_delete script to upload the deleted keys. The MobiLink server then deletes rows from the consolidated pool of values that a user has deleted from his personal value pool in the remote database.

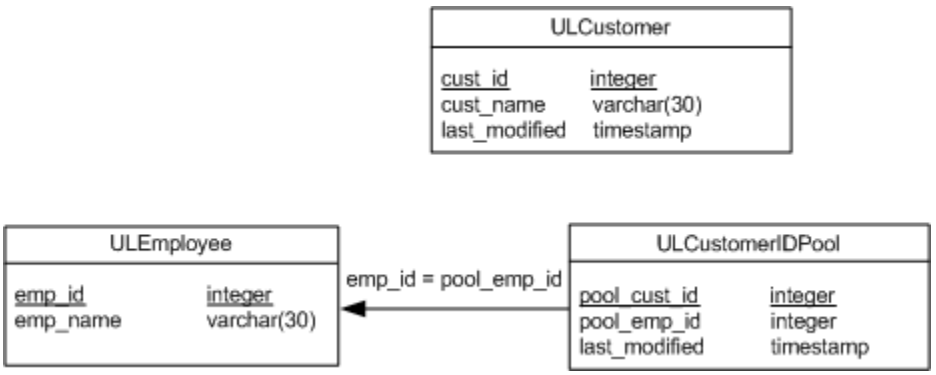
6. Write an end_upload script to call the stored procedure that maintains the pool of values. Doing so has the effect of adding more values to the user's pool to replace those deleted during upload.

Example

The CustDB sample application allows remote users to add customers. It is essential that each new row has a unique primary key value, and yet each remote database is disconnected when data entry is occurring.

The ULCustomerIDPool holds a list of primary key values that can be used by each remote database. In addition, the ULCustomerIDPool_maintain stored procedure tops up the pool as values are used up. The maintenance procedures are called by a table-level end_upload script, and the pools at each remote database are maintained by download_cursor and upload_delete scripts.

1. The ULCustomerIDPool table in the consolidated database holds the pool of new customer identification numbers. It has no direct link to the ULCustomer table.



2. The ULCustomerIDPool_maintain procedure updates the ULCustomerIDPool table in the consolidated database. The following sample code is for a SQL Anywhere consolidated database.

```
CREATE PROCEDURE ULCustomerIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
    DECLARE pool_count INTEGER;

    -- Determine how may ids to add to the pool
    SELECT COUNT(*) INTO pool_count
    FROM ULCustomerIDPool
    WHERE pool_emp_id = syncuser_id;

    -- Top up the pool with new ids
    WHILE pool_count < 20 LOOP
      INSERT INTO ULCustomerIDPool ( pool_emp_id )
      VALUES ( syncuser_id );
      SET pool_count = pool_count + 1;
    END LOOP;
END
```

This procedure counts the numbers that are currently assigned to the current user, and inserts new rows so that this user has enough customer identification numbers.

This procedure is called at the end of the upload, by the end_upload table script for the ULCustomerIDPool table. The script is as follows:

```
CALL ULCustomerIDPool_maintain( {ml s.username} )
```

3. The download_cursor script for the ULCustomerIDPool table downloads new numbers to the remote database.

```
SELECT pool_cust_id
FROM ULCustomerIDPool
WHERE pool_emp_id = {ml s.username}
AND last_modified >= {ml s.last_table_download}
```

4. To insert a new customer, the application using the remote database must select an unused identification number from the pool, delete this number from the pool, and insert the new customer information using this identification number. The following embedded SQL function for an UltraLite application retrieves a new customer number from the pool.

```
bool CDemoDB::GetNextCustomerID( void )
/************************************/
{
    short ind;

    EXEC SQL SELECT min( pool_cust_id )
    INTO :m_CustID:ind FROM ULCustomerIDPool;
    if( ind < 0 ) {
        return false;
    }
    EXEC SQL DELETE FROM ULCustomerIDPool
    WHERE pool_cust_id = :m_CustID;
    return true;
}
```

Handling conflicts

Conflicts can arise during the upload of rows to the consolidated database. If two users modify the same row on different remote databases or if one user modifies a row in the consolidated database and one user modifies the same row in a remote database, a conflict is detected when the second of the rows arrives at the MobiLink server.

By default,

- If an attempt to insert a row finds that the row has already been inserted, an error is generated.
- If an attempt to delete a row finds that the row has already been deleted, the second attempt to delete is ignored.

If you need different behavior, you can implement it by defining one or more of the upload events that are described in this section.

About conflicts

Caution
Never update primary keys in synchronized tables. Updating primary keys defeats the purpose of a primary key because the key is the only way to identify the same row in different databases (remote and consolidated) and the only way to detect conflicts.

Conflicts are not the same as errors. When conflicts can occur, you should define a process to compute the correct values, or at least to log the conflict. Conflict handling is an integral part of a well-designed application.

During the download stage of a synchronization, no conflicts arise in the remote database. If a downloaded row contains a new primary key, the values are inserted into a new row. If the primary key matches that of a pre-existing row, the values in the row are updated.

Example

User1 starts with an inventory of ten items, and then sells three and updates the Remote1 inventory value to seven items. User2 sells four items and updates the Remote2 inventory to six. When Remote1 synchronizes, the consolidated database is updated to seven. When Remote2 synchronizes, a conflict is detected because the value of the inventory is no longer ten. To resolve this conflict programmatically, you need three row values:

1. The current value in the consolidated database.
2. The new row value that Remote2 uploaded.
3. The old row value that Remote2 obtained during the last synchronization.

In this case, the business logic could use the following to calculate the new inventory value and resolve the conflict:

```
current consolidated - (old remote - new remote)
-> 7 - (10-6) = 3
```

For other examples of how to handle conflicts, see:

- "Synchronizing products in the Contact sample" [*MobiLink - Getting Started*]

Detecting conflicts

When a MobiLink client sends an updated row to the MobiLink server, it includes not only the new updated values (the post-image), but also a copy of the old row values (the pre-image) obtained either in the last download or from the row values existing before the first upload of this row. When the pre-image does not match the current values in the consolidated database, a conflict is detected.

There are several scripts provided to detect conflicts. The MobiLink server detects conflicts only if one of the following scripts is applied:

- An upload_fetch or upload_fetch_column_conflict script.
- An upload_new_row_insert or upload_old_row_insert script.
- An upload_update script that includes all non-primary key columns in the WHERE clause (deprecated).

Detecting conflicts with upload_fetch or upload_fetch_column_conflict scripts

If you define an upload_fetch or upload_fetch_column_conflict script for a table, the MobiLink server compares the pre-image of an uploaded update to the values of the row returned by the script with the same primary key values. The MobiLink server detects a conflict if values in the pre-image do not match the current consolidated values. The server calls the upload_old_row_insert and upload_new_row_insert scripts followed by the resolve_conflict script when a conflict is detected.

Note
An error occurs if the upload_old_row_insert and upload_new_row_insert scripts are not defined during a conflict. Define these scripts as ignored using the --{ml_ignore} statement if they are not required for the synchronization table.

The difference between upload_fetch and upload_fetch_column_conflict scripts is in the criterion the MobiLink server uses to detect a conflict. With an upload_fetch script any difference between the fetched row and pre-image row is treated as a conflict. With an upload_fetch_column_conflict script, only the columns updated by the remote database are compared between the fetched row and the pre-image row. In other words, upload_fetch provides row-based conflict detection, and upload_fetch_column_conflict provides column-based conflict detection.

The upload_fetch script selects a single row of data from a consolidated database table corresponding to the row being updated. There are two ways to use this script. The first way is to select the row with the same primary key(s) and same column values as the uploaded pre-image. If no row is returned, MobiLink server detects a conflict. This method of using the script has the following syntax:

SELECT *pk1*, *pk2*, ...*col1*, *col2*, ...
FROM *table-name*
WHERE *pk1* **= {ml r.***pk1***} AND** *pk2* **= {ml r.***pk2***}** ...
  AND *col1* **= {ml o.***col1***} AND** *col2* **= {ml o.***col2***}** ...

Note
This method of conflict detection can not be used with synchronized tables that have large binary columns such as BLOB and CLOB.

The second way is to select the row with the same primary key, letting MobiLink server compare the fetched row against the uploaded pre-image. If any columns differ, the MobiLink server detects a conflict. This approach works with all synchronizable column types:

SELECT *pk1*, *pk2*, ...*col1*, *col2*, ...
FROM *table-name*
WHERE *pk1* **= {ml r.***pk1***} AND** *pk2* **= {ml r.***pk2***}** ...

See "upload_fetch table event" on page 454.

The upload_fetch_column_conflict event is the same as upload_fetch, except that with it the MobiLink server only detects a conflict for a row when the same column was updated on the remote database and the consolidated database since the last synchronization. Different users can update the same row without generating a conflict, as long as they don't update the same column. The upload_fetch_column_conflict event can only be applied to synchronization tables that have no BLOBs.

When using an upload_fetch_column_conflict script and no conflict is detected, the row values passed to your upload_update script come from either the remote database's upload or the current consolidated values from your upload_fetch_column_conflict script. The remote database's value is used for columns that were updated on the remote database, otherwise the current consolidated value is used. In other words, only the columns that were updated on the remote are updated in the consolidated.

See "upload_fetch_column_conflict table event" on page 469.

You can have only one upload_fetch or upload_fetch_column_conflict script for each table in the remote database.

Locking the row on the consolidated database

A row on the consolidated database might change after the upload_fetch script detects a conflict and before the conflict resolution is completed. To avoid this problem, which could result in incorrect data, you can implement the upload_fetch or upload_fetch_column_conflict scripts with a row lock.

In SQL Anywhere consolidated databases, you can use either the UPDLOCK or HOLDLOCK keywords, but UPDLOCK is better for concurrency. For example:

```
SELECT column-names from table-name WITH (UPDLOCK)
  WHERE where-clause
```

For Microsoft SQL Server, use HOLDLOCK. For example,

```
SELECT column-names FROM table-name WITH (HOLDLOCK)
  WHERE where-clause
```

For Adaptive Server Enterprise, use HOLDLOCK. For example,

```
SELECT column-names FROM table-name
  HOLDLOCK
  WHERE where-clause
```

Example

You define an upload_fetch script. The MobiLink server uses the script to retrieve the current row in the consolidated database and compares this row to the pre-image of the updated row. If the two rows contain identical values, there is no conflict. If the two rows differ, then a conflict is detected and MobiLink calls the upload_old_row_insert and upload_new_row_insert scripts, followed by resolve_conflict.

See "Resolving conflicts with resolve_conflict scripts" on page 107.

Detecting conflicts with upload_new_row_insert or upload_old_row_insert scripts

If you define either an upload_new_row_insert or upload_old_row_insert script for a table without defining the upload_fetch and upload_fetch_column_conflict scripts, the MobiLink server uses the number of rows affected by the upload updates to determine if a conflict has occurred.

If the MobiLink server is attempting to upload multiple rows during a conflict and the number of affected rows is less than the number of rows applied to the consolidated database, then the server uploads each

row individually to re-apply the upload updates. However, if the MobiLink server is attempting to upload a single row and the number of affected rows is zero after the upload update, then the server treats the update as a conflict update. For more information about uploading multiple rows, see "-s mlsrv12 option" on page 64.

Note
The number of affected rows count is not always reliable. The MobiLink server could trigger an incorrect conflict update and call the conflict resolution scripts mistakenly if the affected row count is not correct.

See also

- "Detecting conflicts with upload_fetch or upload_fetch_column_conflict scripts" on page 104
- "upload_new_row_insert table event" on page 505
- "upload_old_row_insert table event" on page 507

Detecting conflicts with upload_update scripts (deprecated)

To use the upload_update script to detect conflicts, include all columns in the WHERE clause:

UPDATE *table-name*
SET *col1* **= {ml r.***col1***},** *col2* **= {ml r.** *col2* **}** ...
WHERE *pk1* **= {ml o.***pk1***} AND** *pk2* **= {ml o.***pk2***}** ...
  AND *col1* **= {ml o.***col1***} AND** *col2* **= {ml o.***col2***}** ...

In this statement, col1, col2 and so on are the non-primary key columns, while pk1, pk2 and so on are primary key columns. The values passed to the second set of non-primary key columns (o.) are the pre-image (or old values) of the updated row. The WHERE clause compares old values uploaded from the remote to current values in the consolidated database. If the values do not match, the update is ignored, preserving the values already on the consolidated database.

See "upload_update table event" on page 519.

The upload_update script is used for conflict detection only if no conflict is detected by upload_fetch or upload_fetch_column_conflict.

Caution
The MobiLink server cannot accurately assess how many rows were changed by an upload script. For this reason, conflict detection using the upload_update script is deprecated.

Scenario 1 (deprecated)

You define scripts for the following events: upload_update, upload_old_row_insert, upload_new_row_insert, and resolve_conflict.

You define the following upload_update script:

```
UPDATE product
SET name={ml r.name}, description={ml r.description}
WHERE id={ml r.id}
```



```
AND name={ml o.name}
AND description={ml o.description}
```

MobiLink performs the update and then checks to see how many rows were modified. If no rows were modified, then MobiLink has detected a conflict: no row in the consolidated database matches the pre-image row. MobiLink calls the upload_old_row_insert and upload_new_row_insert scripts, followed by resolve_conflict.

See "Resolving conflicts with resolve_conflict scripts" on page 107.

Scenario 2

This method is not deprecated because it does not involve the MobiLink server directly doing any detection or resolution.

You do not define scripts for upload_old_row_insert, upload_new_row_insert, and resolve_conflict. Instead, you create a stored procedure to handle the conflict detection and resolution and you call it in the upload_update script.

See "Resolving conflicts with upload_update scripts" on page 109.

Resolving conflicts

You have several options for resolving conflicts:

- Resolve conflicts as they occur using temporary or permanent tables and a resolve_conflict script.

  See "Resolving conflicts with resolve_conflict scripts" on page 107.

- Resolve conflicts as they occur using an upload_update script.

  See "Resolving conflicts with upload_update scripts" on page 109.

- Resolve all conflicts at once using a table's end_upload script.

  See "end_upload table event" on page 374.

Resolving conflicts with resolve_conflict scripts

When the MobiLink server detects a conflict using an upload_fetch script, the following events take place.

- The MobiLink server inserts old row values uploaded from the remote database as defined by the upload_old_row_insert script. Typically, the old values are inserted into a temporary table.

  See "upload_old_row_insert table event" on page 507.

- The MobiLink server inserts the new row values uploaded from the remote database as defined by the upload_new_row_insert script. Typically, the new values are inserted into a temporary table.

  See "upload_new_row_insert table event" on page 505.

- The MobiLink server executes the resolve_conflict script. In this script you can either call a stored procedure, or define a sequence of steps to resolve the conflict using the new and old row values.

For more information, see "resolve_conflict table event" on page 422.

Example

In the following example, you create scripts for six events and then you create a stored procedure.

- In the begin_synchronization script, you create two temporary tables called contact_new and contact_old. (You could also do this in the begin_connection script.)
- The upload_fetch script detects the conflict.
- When there is a conflict, the upload_old_row_insert and upload_new_row_insert scripts populate the two temporary tables with the new and old data uploaded from the remote database.
- The resolve_conflict script calls the stored procedure MLResolveContactConflict to resolve the conflict.

Event	Script

begin_synchronization	`CREATE TABLE #contact_new(`
	`  id  INTEGER,`
	`  location CHAR(36),`
	`  contact_date DATE);`
	`CREATE TABLE #contact_old(`
	`  id  INTEGER,`
	`  location CHAR(36),`
	`  contact_date DATE)`

upload_fetch	`SELECT id, location, contact_date`
	`  FROM contact`
	`  WHERE id = {ml r.id}`

upload_old_row_insert	`INSERT INTO #contact_new( id, location, contact_date )` `  VALUES ( {ml r.id}, {ml r.location}, {ml r.contact_date} )`
upload_new_row_insert	`INSERT INTO #contact_old( id, location, contact_date )` `  VALUES ( {ml r.id}, {ml r.location}, {ml r.contact_date} )`
resolve_conflict	`CALL MLResolveContactConflict( )`
end_synchronization	`DROP TABLE #contact_new;` `DROP TABLE #contact_old`

The stored procedure MLResolveContactConflict is as follows:

```
CREATE PROCEDURE MLResolveContactConflict( )
BEGIN
  --update the consolidated database only if the new contact date
  --is later than the existing contact date
  UPDATE contact c
    SET c.contact_date = cn.contact_date
```



```
      FROM #contact_new cn
      WHERE c.id = cn.id
        AND cn.contact_date > c.contact_date;
  --cleanup
  DELETE FROM #contact_new;
  DELETE FROM #contact_old;
END
```

Resolving conflicts with upload_update scripts

Instead of using the resolve_conflict script for conflict resolution, you can call a stored procedure in the upload_update script. With this technique, you must both detect and resolve conflicts programmatically.

The stored procedure must accept all columns, including both the new (post-image) and old (pre-image) values.

The upload_update script could be as follows:

```
{CALL UpdateProduct(
   {ml o.id}, {ml o.name}, {ml o.desc}, {ml r.name}, {ml r.desc}
)
}
```

The UpdateProduct stored procedure could be:

```
CREATE PROCEDURE UpdateProduct(
  @id INTEGER,
  @preName VARCHAR(20),
  @preDesc VARCHAR(200),
  @postName VARCHAR(20),
  @postDesc VARCHAR(200) )
BEGIN
    UPDATE product
     SET name = @postName, description = @postDesc
     WHERE id = @id
       AND name = @preName
       AND description = @preDesc
    IF @@rowcount=0 THEN
        // A conflict occurred: handle resolution here.
    END IF
END
```

This approach is often easier to maintain than resolving conflicts with resolve_conflict scripts because there is only one script to maintain and all the logic is contained in one stored procedure. However, the code of the stored procedure may be complicated if the tables columns are nullable or if they contain BLOBs or CLOBs. Also, some RDBMSs that are supported MobiLink consolidated databases have limitations on the size of values that can be passed to stored procedures.

See:

- “Detecting conflicts with upload_update scripts (deprecated)” on page 106
- “upload_update table event” on page 519
- “Resolving conflicts with resolve_conflict scripts” on page 107

Example

The following stored procedure, sp_update_my_customer, contains logic for conflict detection and resolution. It accepts old column values and new column values. This example uses SQL Anywhere features. The script could be implemented as follows.

```
{CALL sp_update_my_customer(
   {ml o.cust_1st_pk},
   {ml o.cust_2nd_pk},
   {ml o.first_name},
   {ml o.last_name},
   {ml o.nullable_col},
   {ml o.last_modified},
   {ml r.first_name},
   {ml r.last_name},
   {ml r.nullable_col},
   {ml r.last_modified}
)}
CREATE PROCEDURE sp_update_my_customer(
     @cust_1st_pk        INTEGER,
     @cust_2nd_pk        INTEGER,
     @old_first_name     VARCHAR(100),
     @old_last_name      VARCHAR(100),
     @old_nullable_col   VARCHAR(20),
     @old_last_modified  DATETIME,
     @new_first_name     VARCHAR(100),
     @new_last_name      VARCHAR(100),
     @new_nullable_col   VARCHAR(20),
     @new_last_modified  DATETIME
)
BEGIN
DECLARE @current_last_modified DATETIME;
// Detect a conflict by checking the number of rows that are
// affected by the following update. The WHERE clause compares
// old values uploaded from the remote to current values in
// the consolidated database. If the values match, there is
// no conflict. The COALESCE function returns the first non-
// NULL expression from a list, and is used in this case to
// compare values for a nullable column.


UPDATE my_customer
SET first_name                  = @new_first_name,
 last_name                      = @new_last_name,
 nullable_col                   = @new_nullable_col,
 last_modified                  = @new_last_modified

WHERE cust_1st_pk               = @cust_1st_pk
 AND cust_2nd_pk                = @cust_2nd_pk
 AND first_name                 = @old_first_name
 AND last_name                  = @old_last_name
 AND COALESCE(nullable_col, '') = COALESCE(@old_nullable_col, '')
 AND last_modified              = @old_last_modified;
...
// Use the @@rowcount global variable to determine
// the number of rows affected by the update. If @@rowcount=0,
// a conflict has occurred. In this example, the database with
// the most recent update wins the conflict. If the consolidated
// database wins the conflict, it retains its current values
// and no action is taken.

IF( @@rowcount = 0 ) THEN
// A conflict has been detected. To resolve it, use business
```



```
// logic to determine which values to use, and update the
// consolidated database with the final values.

  SELECT last_modified INTO @current_last_modified
     FROM my_customer WITH( HOLDLOCK )
     WHERE cust_1st_pk=@cust_1st_pk
        AND cust_2nd_pk=@cust_2nd_pk;

     IF( @new_last_modified > @current_last_modified ) THEN
     // The remote has won the conflict: use the values it
     // uploaded.

       UPDATE my_customer
       SET first_name    = @new_first_name,
         last_name     = @new_last_name,
         nullable_col  = @new_nullable_col,
         last_modified = @new_last_modified
       WHERE cust_1st_pk  = @cust_1st_pk
          AND cust_2nd_pk   = @cust_2nd_pk;

     END IF;
END IF;
END;
```

See:

- "COALESCE function [Miscellaneous]" [*SQL Anywhere Server - SQL Reference*]
- @@rowcount in "Global variables" [*SQL Anywhere Server - SQL Reference*]

Forced conflicts (Deprecated)

Forced conflict resolution is a special technique that forces every uploaded row to be treated as if it were a conflict.

The MobiLink server uses forced conflict resolution when the upload_insert, upload_update, and upload_delete script are all undefined. In this mode of operation, the MobiLink server attempts to insert all uploaded rows from that table using the statements defined by the upload_old_row_insert and upload_new_row_insert scripts. In essence, all uploaded rows are then treated as conflicts. You can write stored procedures or scripts to process the uploaded values in any way you want.

Normal conflict-resolution occurs when an upload_fetch or upload_fetch_column_conflict script is defined. Without any of the upload_insert, upload_update, or upload_delete scripts, the normal conflict-resolution procedure is bypassed. This technique has two principal uses.

- **Arbitrary conflict detection and resolution** The automatic mechanism only detects errors when updating a row, and only then when the old values do not match the present values in the consolidated database.

  You can capture the raw uploaded data using the upload_old_row_insert and upload_new_row_insert scripts, then process the rows as you see fit.

- **Performance** When the upload_insert, upload_update, and upload_delete are not defined, the MobiLink server is relieved of its normal conflict-detection tasks, which involve querying the

consolidated database one row at a time. Instead, it needs only to insert the raw uploaded information using the statements defined by the upload_old_row_insert and upload_new_row_insert scripts. Performance is improved because the MobiLink server is not fetching rows across the network.

See also

- "Forced conflict statistics" on page 172

Handling deletes

When rows are deleted from the consolidated database, there needs to be a record of the row so it can be explicitly selected by a download_delete_cursor and removed from any remote databases that have the row. Two ways to do this are by using logical deletes or shadow tables.

- **Logical deletes** With this method, the row is not deleted. Data that is no longer required is marked as inactive in a status column. The WHERE clause of the download_cursor and download_delete_cursor and most application queries must refer to the status of the row.

  This technique is used in the CustDB sample application, in which the ULEmpCust.action column holds a D for Delete. The scripts use this value to delete records from the remote database, and delete records from the consolidated database at the end of the synchronization. CustDB also uses this technique for the ULOrder table, and the Contact sample uses the technique on the Customer, Contact, and Product tables.

  The MobiLink synchronization model support for logical deletes assumes that a logical delete column is only on the consolidated database and not on the remote. When copying a consolidated schema to a new remote schema, leave out any columns that match the logical delete column in the model's synchronization settings. For a new model, the default column name is *deleted*.

  To add the logical delete column name to the remote schema:

  1. In the **Create Synchronization Model Wizard**, on the **Download Deletes** page, choose **Use logical deletes**.
  2. Rename the logical delete column so that it does not match any column names in the consolidated.
  3. When the wizard is finished, update the remote schema and keep the default table selection. The logical delete column name appears in the schema change list and be added to remote schema.

  Note
  You need to set the column mapping for the remote's logical delete column to the consolidated's logical delete column.

- **Shadow tables** With this method, you create a shadow table that stores the primary key values of deleted rows. When a row is deleted, a trigger can populate the shadow table. The download_delete_cursor can use the shadow table to remove rows from remote databases. The shadow table only needs to contain the primary key columns from the real table.

See "Writing download_delete_cursor scripts" on page 293.

Temporarily stopping the synchronization of deletes

SQL Anywhere automatically logs any changes to tables or columns that are part of a publication with a synchronization subscription. These changes are normally uploaded to the consolidated database during the next synchronization.

You may, however, want to delete rows from synchronized data and not have those changes uploaded. You can do this using STOP SYNCHRONIZATION DELETE.

When a STOP SYNCHRONIZATION DELETE statement is executed, none of the delete operations subsequently executed on that connection are synchronized. The effect continues until a START SYNCHRONIZATION DELETE statement is executed. The effects do not nest; that is, subsequent executions of STOP SYNCHRONIZATION DELETE after the first have no additional effect.

This feature can be used to make unusual corrections, but should be used with caution as it effectively disables part of the automatic synchronization functionality. This technique is a practical alternative to deleting the necessary rows using a download_delete_cursor script.

To temporarily disable upload of deletes made through a connection

1. Issue the following statement to stop automatic logging of deletes.

   ```
   STOP SYNCHRONIZATION DELETE
   ```

2. Delete rows from the synchronized table(s), as required, using the DELETE statement. Commit these changes.

3. Restart logging of deletes using the following statement.

   ```
   START SYNCHRONIZATION DELETE
   ```

The deleted rows are not sent up to the MobiLink server and are not deleted from the consolidated database.

See also

- SQL Anywhere clients: "STOP SYNCHRONIZATION DELETE statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- UltraLite clients: "STOP SYNCHRONIZATION DELETE statement [UltraLite] [UltraLiteJ]" [*UltraLite - Database Management and Reference*]

Handling failed downloads

Using non-blocking download acknowledgement

Bookkeeping information about what is downloaded must be maintained in the nonblocking download acknowledgement transaction. This information should be updated in the publication_nonblocking_download_ack or nonblocking_download_ack scripts which is called after the remote database successfully applies the download.

If a failure occurs or SendDownloadAck is OFF, these non-blocking download acknowledgement scripts are not called and the download timestamp in the consolidated database is not updated. When testing your synchronization scripts you should artificially cause failed downloads to ensure that your scripts can handle a failed download.

Using blocking download acknowledgement

Support for blocking download acknowledgement has been discontinued. All download acknowledgements are handled as non-blocking.

See "SendDownloadAck (sa) extended option" [*MobiLink - Client Administration*] and "Send Download Acknowledgement synchronization parameter" [*UltraLite - Database Management and Reference*].

Resuming failed downloads

Download failure is caused by a communication error during the download or a remote user cancelling the download. The MobiLink server holds download data that has not been received by the client so it may be used for a restartable. The server does not release download data until one of the following occurs:

- The user successfully completes the download.
- The user comes back with a new synchronization request without attempting to resume.
- The cache is needed for new downloads. The oldest unsuccessful download is cleared first.

MobiLink has functionality that can assist with download failure recovery, and prevent retransmission of the entire download. This functionality has separate implementations for SQL Anywhere and UltraLite remote databases. See "-ds mlsrv12 option" on page 46.

Note
For a resumable download attempt to succeed, the remote database must connect to the same MobiLink server. If the remote database connects to a different MobiLink server or if the download has been pushed out of the same MobiLink server, the attempted resume will fail.

When should you resume failed downloads?

The need for resumable downloads increases as network quality deteriorates and download sizes increase. If you only do small synchronizations or if you synchronize on a LAN or WLAN, then you likely do not need to resume downloads.

SQL Anywhere remote databases

When synchronization fails during a download, the downloaded data is not applied to the remote database, however, the successfully transmitted portions of the download are stored in a temporary file on the remote device. Dbmlsync uses this file to avoid lengthy retransmission of data, and to recover from download failure.

There are three ways to implement this functionality. For all options, dbmlsync aborts and the resumed download fails if there is any new data to be uploaded.

- **-dc** After a download fails, use -dc the next time you start dbmlsync to resume the download. If part of the failed download was transmitted, the MobiLink server only transmits the remainder of the download.

  For more information, see "-dc dbmlsync option" [*MobiLink - Client Administration*].

- **ContinueDownload (cd) extended option** When used on the dbmlsync command line, the cd extended option works just like the -dc option. You can also store this option in the database, or use sp_hook_dbmlsync_set_extended_options to set this option in a single synchronization.

  See "ContinueDownload (cd) extended option" [*MobiLink - Client Administration*] and "sp_hook_dbmlsync_set_extended_options" [*MobiLink - Client Administration*].

- **sp_hook_dbmlsync_end hook** You can use the restart parameter to cause a download to resume. You know a download is resumable if the restartable download parameter is set to true. You can also create logic in the hook to resume a download if a download file exists and is a certain size, by using the restartable download size.

  See "sp_hook_dbmlsync_end" [*MobiLink - Client Administration*].

UltraLite remote databases

You can control the behavior of UltraLite applications following a failed download as follows:

- If you set the Keep Partial Download synchronization parameter to true when you synchronize, and the download fails before completion, then UltraLite applies that portion of the changes that were downloaded. UltraLite also sets the Partial Download Retained synchronization parameter to true.

  The UltraLite database may be in an inconsistent state at this point. Depending on your application, you may want to ensure that synchronization completes successfully or is rolled back before you allow changes to the data. See "Keep Partial Download synchronization parameter" [*UltraLite - Database Management and Reference*], and "Partial Download Retained synchronization parameter" [*UltraLite - Database Management and Reference*].

- To resume the download, set the Resume Partial Download synchronization parameter to true and synchronize again. See "Resume Partial Download synchronization parameter" [*UltraLite - Database Management and Reference*].

  The restarted synchronization does not perform an upload, and downloads only those changes that would have been downloaded by the failed download. That is, it completes the failed download but does not synchronize changes made since the previous attempt. To get those changes, you need to synchronize again once the failed download has completed, or call Rollback Partial Download and synchronize with Resume Partial Download set to false.

  When you restart the download, many of the synchronization parameters from the failed synchronization are used again automatically. For example, the publications parameter is ignored: the synchronization downloads those publications requested on the initial download. The only parameters

that must be set are the Resume Partial Download parameter (which must be set to true) and the User Name parameter. In addition, settings for the following parameters are obeyed, if set:

- Keep Partial Download
- DisableConcurrency
- Observer
- User Data

- To roll back the changes from the failed download without resuming synchronization, call the appropriate method or function to roll back the changes. This function is ULRollbackPartialDownload function for embedded SQL. For UltraLite components, it is a method on the Connection object.

  - **UltraLite.NET** See "RollbackPartialDownload method" [*UltraLite - .NET Programming*].

  - **Embedded SQL** See "ULRollbackPartialDownload method" [*UltraLite - C and C++ Programming*].

  You may want to roll back the changes from a failed download if synchronization cannot be completed, for example if the server or network is unavailable, and if you want to maintain a consistent set of data while letting the end user continue to work.

For more information about communications errors, see "Error Messages".

Note
If the send_download_ack synchronization parameter is set to true, the setting is ignored for the resumed download.

Download acknowledgement

Download acknowledgement is an optional component of synchronization where the client immediately informs MobiLink server when the download is successfully applied at the remote database. It is recommended for deployments whose business logic must act as soon as possible when remote receipt of a download is received. It is not required to ensure that your data is received at the remote.

There are two modes of download acknowledgement: blocking, which has been discontinued, and non-blocking. All download acknowledgements are now handled as non-blocking.

To use download acknowledgements, there are settings on both the client and server.

On the client, you specify download acknowledgement with the dbmlsync extended option SendDownloadAck or the UltraLite synchronization parameter Send Download Acknowledgement.

On the server, there are two connection events that you can use to record the last successful download time in your consolidated database when using non-blocking download acknowledgement, publication_nonblocking_download_ack connection event and nonblocking_download_ack connection event.

Note
Download acknowledgement can not be used with resumable downloads. See "Resuming failed downloads" on page 114.

See also

- "publication_nonblocking_download_ack connection event" on page 414
- "nonblocking_download_ack connection event" on page 410
- dbmlsync: "SendDownloadAck (sa) extended option" [*MobiLink - Client Administration*]
- UltraLite: "Send Download Acknowledgement synchronization parameter" [*UltraLite - Database Management and Reference*]

Downloading a result set from a stored procedure call

You can download a result set from a stored procedure call. For example, you might currently have a download_cursor for the following table:

```
CREATE TABLE MyTable (
    pk INTEGER PRIMARY KEY NOT NULL,
    col1 VARCHAR(100) NOT NULL,
    col2 VARCHAR(20) NOT NULL
    employee VARCHAR(100) NOT NULL
    last_modified TIMESTAMP NOT NULL DEFAULT TIMESTAMP
)
```

The download_cursor table script might look as follows:

```
SELECT pk, col1, col2
   FROM MyTable
      WHERE last_modified >= {ml s.last_table_download}
      AND employee = {ml s.username}
```

If you want your downloads to MyTable to use more sophisticated business logic, you can now create your script as follows, where DownloadMyTable is a stored procedure taking two parameters (last-download timestamp and MobiLink user name) and returning a result set. (This example uses an ODBC calling convention for portability.):

```
{call DownloadMyTable( {ml s.last_table_download}, {ml s.username} )}
```

The following are some simple examples for each supported consolidated database. Consult the documentation for your consolidated database for full details.

The following example works with SQL Anywhere, Adaptive Server Enterprise, and Microsoft SQL Server.

```
CREATE PROCEDURE DownloadMyTable
    @last_dl_ts DATETIME,
    @u_name VARCHAR( 128 )
AS
BEGIN
     SELECT pk, col1, col2
       FROM MyTable
          WHERE last_modified >= @last_dl_ts
          AND employee = @u_name
END
```

For Oracle, the result set can be returned by a REF CURSOR defined in a stored procedure. However, when using the iAnywhere Solutions 12 - Oracle ODBC driver, the REF CURSOR parameter should be defined as the last one in the parameter list of the stored procedure. The REF CURSOR parameter can be defined as OUT or IN OUT. The following stored procedure works with Oracle.

```
create or replace procedure DownloadMyTable(
    v_last_dl_ts IN TIMESTAMP,
    v_user_name IN VARCHAR,
    v_ref_crsr OUT SYS_REFCURSOR ) As
Begin
    Open v_ref_crsr For
        select pk, col1, col2
            from MyTable
                where last_modified >= v_last_dl_ts
                and employee = v_user_name;
End DownloadMyTable;
```

Next, use the ml_add_table_script stored procedure to define a call to DownloadMyTable as the download_cursor script for the synchronization table MyTable:

```
CALL ml_add_table_script(
  'v1',
  'MyTable',
  'download_cursor',
  '{CALL DownloadMyTable(
     {ml s.last_table_download},{ml s.username} )}'
);
```

For Oracle, note that the DownloadMyTable stored procedure only takes two parameters, not three, and the MobiLink server fetches the result set through the REF CURSOR. The REF CURSOR is defined as the last parameter in the stored procedure definition.

The following example works with IBM DB2 LUW.

```
CREATE PROCEDURE DownloadMyTable(
   IN last_dl_ts TIMESTAMP,
   IN u_name VARCHAR( 128 ) )
        LANGUAGE SQL
        MODIFIES SQL DATA
        COMMIT ON RETURN NO
        DYNAMIC RESULT SETS 1

        BEGIN
            DECLARE C1, cursor WITH RETURN FOR
                SELECT pk, col1, col2 FROM MyTable
                WHERE last_modified >= last_dl_ts AND employee = u_name;
            OPEN C1;
        END;
```

Uploading data from self-referencing tables

Some tables are self-referencing. For example, an employee table may contain a column that lists employees and a column that lists the manager of each employee, and there may be a hierarchy of managers managing managers. These tables can pose a challenge to synchronization because the MobiLink default behavior is to coalesce all data updates on the remote database, which is efficient but which loses the order of transactions.

There are two techniques for handling this situation:

- If you are using a SQL Anywhere remote database, you can use the dbmlsync -tu option to specify that each transaction on the remote should be sent as a separate transaction. See "-tu dbmlsync option" [*MobiLink - Client Administration*].
- Add a mapping table, mapping employees to managers, so the order of transactions in the formerly self-referencing table no longer matters.

MobiLink isolation levels

MobiLink connects to a consolidated database at the most optimal isolation level it can, given the isolation levels enabled on the RDBMS. The default isolation levels are chosen to provide the best performance while ensuring data consistency.

In general, MobiLink uses the isolation level SQL_TXN_READ_COMMITTED for uploads, and if possible, it uses snapshot isolation for downloads. If snapshot isolation is not available MobiLink uses SQL_TXN_READ_COMMITTED. A download using SQL_TXN_READ_COMMITTED isolation has the potential to block until another transaction completes. Such blocking can significantly decrease the throughput of synchronizations. Snapshot isolation eliminates the problem of downloads being blocked until transactions are closed on the consolidated database assuming the download performs no updates, which is highly recommended.

Snapshot isolation can result in duplicate data being downloaded (if, for example, a long-running transaction causes the same snapshot to be used for a long time), but MobiLink clients automatically handle this, so the only penalty is transmission time and the processing effort at the remote. Nevertheless, avoiding long-running transactions is recommended.

Isolation level 0 (READ UNCOMMITTED) is generally unsuitable for synchronization and can lead to inconsistent data.

The isolation level is set immediately after a connection to the consolidated database occurs. Some other connection setup also occurs at that time, and then the transaction is committed. The COMMIT is required by most RDBMSs so that the isolation level (and perhaps other settings) can take effect.

SQL Anywhere version 10 and later consolidated databases

SQL Anywhere versions 10 and later support snapshot isolation. By default, MobiLink uses the SQL_TXN_READ_COMMITTED isolation level for uploads, and snapshot isolation for downloads.

MobiLink can only use snapshot isolation if you enable it in your SQL Anywhere consolidated database. If snapshot isolation is not enabled, MobiLink uses the default SQL_TXN_READ_COMMITTED.

Enabling a database to use snapshot isolation can affect performance because copies of all modified rows must be maintained, regardless of the number of transactions that use snapshot isolation. See "Enabling snapshot isolation" [*SQL Anywhere Server - SQL Usage*].

You can enable snapshot isolation for upload with the mlsrv12 -esu option, and disable snapshot isolation with the mlsrv12 -dsd option. If you need to change the MobiLink default isolation level in a connection

script, you should do so in the begin_upload or begin_download scripts. If you change the default isolation level in the begin_connection script, your setting may be overridden at the start of the upload and download transactions.

See "-esu mlsrv12 option" on page 48 and "-dsd mlsrv12 option" on page 46.

SQL Anywhere versions earlier than version 10 consolidated databases

If you are using a version of SQL Anywhere earlier than version 10, the default MobiLink isolation level is SQL_TXN_READ_COMMITTED. You can change the default for the entire MobiLink session in the begin_connection script, or change it for the upload and download in the begin_upload and begin_download scripts, respectively.

Adaptive Server Enterprise consolidated databases

For Adaptive Server Enterprise, the default MobiLink isolation level is SQL_TXN_READ_COMMITTED. You can change the default for the entire MobiLink session in the begin_connection script, or change it for the upload and download in the begin_upload and begin_download scripts, respectively.

Oracle consolidated databases

Oracle supports snapshot isolation, but calls it READ COMMITTED. By default, MobiLink uses the snapshot/READ COMMITTED isolation level for upload and download.

You can change the default for the entire MobiLink session in the begin_connection script, or change it for the upload and download in the begin_upload and begin_download scripts, respectively.

For the MobiLink server to be able to make the most effective use of snapshot isolation, the Oracle account used by the MobiLink server must have permission for the GV_$TRANSACTION Oracle system view. If it does not, a warning is issued and rows may be missed on download. Only SYS can grant this access. The Oracle syntax for granting this access is:

```
grant select on SYS.GV$TRANSACTION to user-name;
```

Microsoft SQL Server 2005 and later consolidated databases

Microsoft SQL Server 2005 supports snapshot isolation. By default, MobiLink uses the SQL_TXN_READ_COMMITTED isolation level for uploads, and snapshot isolation for download.

MobiLink can only use snapshot isolation if you enable it in your SQL Server consolidated database. If snapshot is not enabled, MobiLink uses the default SQL_TXN_READ_COMMITTED. See your SQL Server documentation for details.

You can enable snapshot isolation for upload with the mlsrv12 -esu option, and disable snapshot isolation with the mlsrv12 -dsd option. If you need to change the MobiLink default isolation level in a connection script, you should do so in the begin_upload or begin_download scripts. If you change the default isolation level in the begin_connection script, your setting may be overridden at the start of the upload and download transactions.

See "-esu mlsrv12 option" on page 48 and "-dsd mlsrv12 option" on page 46.

To use snapshot isolation on SQL Server, the user ID that you use to connect the MobiLink server to the database must have permission to access the SQL Server system table SYS.DM_TRAN_ACTIVE_TRANSACTIONS. If this permission is not granted, MobiLink uses the default level SQL_TXN_READ_COMMITTED.

If your consolidated database is running on a Microsoft SQL Server that is also running other databases, and if you are using snapshot isolation for uploads or downloads, and if your upload or download scripts do not access any other databases on the server, you should specify the MobiLink server -dt option. This option makes MobiLink ignore all transactions except ones within the current database, and potentially increases throughput and reduces duplication of rows that are downloaded.

See "-dt mlsrv12 option" on page 47.

See also

- "-dsd mlsrv12 option" on page 46
- "-esu mlsrv12 option" on page 48
- "-dt mlsrv12 option" on page 47
- "The synchronization process" [*MobiLink - Getting Started*]
- "Isolation levels and consistency" [*SQL Anywhere Server - SQL Usage*]
- "Snapshot isolation" [*SQL Anywhere Server - SQL Usage*]

MobiLink performance

Performance tips

The following is a list of suggestions to help you get the best performance out of MobiLink.

Test

The following all contribute to the throughput of your synchronization system:

- the type of device running your remote databases
- the schema of remote databases
- the data volume and synchronization frequency of your remotes
- network characteristics (including for HTTP, proxies, web servers, and Relay Servers)
- the hardware where the MobiLink server runs
- your synchronization scripts
- the concurrent volume of synchronizations
- the type of consolidated database you use

- the hardware where your consolidated database runs
- the activity in the consolidated database, including all non-synchronization activity
- the schema of your consolidated database

Testing is extremely important. Before deploying, you should perform testing using the same hardware and network that you plan to use for production. You should also try to test with the same number of remotes, the same frequency of synchronization, and the same data volume. The MobiLink replay tool can help with such testing. See "MobiLink replay utility (mlreplay)" on page 733.

During this testing you should experiment with the following performance tips.

Avoid contention

Avoid contention and maximize concurrency in your synchronization scripts.

For example, suppose a begin_download script increments a column in a table to count the total number of downloads. If multiple users synchronize at the same time, this script would effectively serialize their downloads. The same counter would be better in the begin_synchronization, end_synchronization, or prepare_for_download scripts because these scripts are called just before a commit so any database locks are held for only a short time. An even better approach would be to only count per remote ID and obtain the total later via a query.

See "Contention" on page 126.

For information about the transaction structure of synchronization, see "Transactions in the synchronization process" [*MobiLink - Getting Started*].

Use an optimal number of database worker threads

Use the MobiLink -w option to set the number of MobiLink database worker threads to the smallest number that gives you optimum throughput. You need to experiment to find the best number for your situation.

A larger number of database worker threads **may** improve throughput by allowing more synchronizations to access the consolidated database at the same time, but with it comes an increased potential for contention and blocking.

Keeping the number of database worker threads small reduces the chance of contention in the consolidated database, the number of connections to the consolidated database, and the memory required for optimal caching.

Do not set the -w and -wu options in a production system without first verifying the optimal settings for these options.

See:

- "Number of database worker threads" on page 127
- "-w mlsrv12 option" on page 74
- "-wu mlsrv12 option" on page 75

Use smaller upload transactions

Large uploads can cause large transactions in the consolidated database and large transactions lead to more locks held in a transaction, which increases blocking and contention. This can have a significant adverse impact on both synchronization throughput and the consolidated database's overall throughput. Smaller uploads reduce blocking and contention, and may significantly improve throughput.

In a MobiLink synchronization system with SQL Anywhere remotes, smaller uploads can be sent via dbmlsync in one of two ways:

- Use the -tu dbmlsync option for transactional uploads. Each transaction is sent separately. See "-tu dbmlsync option" [*MobiLink - Client Administration*].
- Use the dbmlsync Increment (inc) extended option for incremental uploads. Each increment contains coalesced transactions. The bigger the increment, generally the more transactions are coalesced into one upload. See "Increment (inc) extended option" [*MobiLink - Client Administration*].

On the server side, the performance can be tuned by using the -tx option to batch a number of transactions from the client together into a single consolidated-side transaction. This option is handy in that once you set the client-side option, you can simply tune -tx without having to change the clients. See "-tx mlsrv12 option" on page 69.

Test and tune these client-side and server-side options for maximum throughput.

Avoid synchronizing unnecessary BLOBs

It is inefficient to include a BLOB in a row that is synchronized frequently while the BLOB remains unchanged. To avoid this, you can create a table that contains BLOBs and a BLOB ID, and reference the ID in the table that needs to be synchronized.

Set maximum number of database connections

Set the maximum number of MobiLink database connections to be your number of synchronization script versions times the number of MobiLink database worker threads, plus one. This reduces the need for MobiLink to close and create database connections. You set the maximum number of connections with the mlsrv12 -cn option.

See "MobiLink database connections" on page 128 and "-cn mlsrv12 option" on page 43.

Have enough physical memory

Ensure that the computer running the MobiLink server has enough physical memory to accommodate the cache in addition to its other memory requirements. Consider moving to a 64-bit platform if the server needs more than a 1.5 GB memory cache.

The number of synchronizations being actively processed is not limited by the number of database worker threads. The MobiLink server can unpack uploads and send downloads for a large number of synchronizations simultaneously. Once a server starts paging to disk, its throughput will fall significantly so it is very important that the MobiLink server has a large enough memory cache to process these synchronizations without paging to disk. Look for warning 10082 in the server log, or the "Cache is full" alert from the SQL Anywhere Monitor for MobiLink to detect when the cache is too small.

The MobiLink server automatically grows or shrinks its memory cache as appropriate. Use the -cmax, -cmin and -cinit options to control the memory cache for the MobiLink server.

See also

- "-cinit mlsrv12 option" on page 41
- "-cmax mlsrv12 option" on page 42
- "-cmin mlsrv12 option" on page 42

Use enough processing power

You should dedicate enough processing power to MobiLink so that the MobiLink server processing is not a bottleneck. Typically the MobiLink server requires significantly less CPU than the consolidated database. However, using Java or .NET row handling adds to the MobiLink server processing requirement. In practice, network limitations or database contention are more likely to be bottlenecks.

Optimize script execution

The performance of your scripts in the consolidated database is an important factor. It may help to create indexes on your tables so that the upload and download cursor scripts can efficiently locate the required rows. However, too many indexes may slow uploads.

When you use the **Create Synchronization Model Wizard** in Sybase Central to create your MobiLink applications, an index is automatically defined for each download cursor when you deploy the model.

Use minimum logging verbosity

Use the minimum logging verbosity that is compatible with your business needs. By default, verbose logging is off, and MobiLink does not write its log to disk. You can control logging verbosity with the -v option, and enable logging to a file with the -o or -ot options.

As an alternative to verbose log files, you can monitor your synchronizations with the MobiLink Monitor. The MobiLink Monitor does not need to be on the same computer as the MobiLink server, and a Monitor connection has a negligible effect on MobiLink server performance. See "MobiLink Monitor" on page 154.

Plan for operating system limitations

Operating systems restrict the number of concurrent connections a server can support over TCP/IP. If this limit is reached, which may occur when over 1000 clients attempt to synchronize at the same time, the operating system may exhibit unexpected behavior, such as unexpectedly closing connections and rejecting additional clients that attempt to connect. To prevent this behavior, either configure the operating system to have a higher TCP/IP connection limit and set the -nc option, or use the -sm option to specify a maximum number of remote connections that is less than the operating system limit.

When a client attempts to synchronize with a MobiLink server that has accepted its maximum number of concurrent synchronizations as specified by the -sm option, the client receives the error code -85 (SQLE_COMMUNICATIONS_ERROR). The client application should handle this error and try to connect again in a few minutes.

See:

- "-sm mlsrv12 option" on page 67
- "-nc mlsrv12 option" on page 52
- "Communication error" [*Error Messages*]

Java or .NET vs. SQL synchronization logic

No significant throughput difference has been found between using Java or .NET synchronization logic vs. SQL synchronization logic. However, Java and .NET synchronization logic have some extra overhead per synchronization and require more memory.

In addition, SQL synchronization logic is executed on the computer that runs the consolidated database, while Java or .NET synchronization logic is executed on the computer that runs the MobiLink server. So, Java or .NET synchronization logic may be desirable if your consolidated database is heavily loaded.

Synchronization using direct row handling imposes a heavier processing burden on the MobiLink server, so you may need more RAM, perhaps more disk space, and perhaps more CPU power, depending on how you implement direct row handling.

Priority synchronization

If you have some tables that you need to synchronize more frequently than others, create a separate publication and subscription for them. When using synchronization models in Sybase Central, you can do this by creating more than one model. You can synchronize this priority publication more frequently than other publications, and synchronize other publications at off-peak times.

Download only the rows you need

Take care to download only the rows that are required, for example by using timestamp synchronization instead of snapshot. Downloading unnecessary rows is wasteful and adversely affects synchronization performance.

Only synchronize when you need to

Overly frequent synchronization can create an unnecessary burden on the MobiLink synchronization system. Carefully decide how often you need to synchronize. Test thoroughly to ensure performance expectations can be within the production environment.

For large uploads, estimate the number of rows

For SQL Anywhere clients, you can significantly improve the speed of uploading a large number of rows by providing dbmlsync with an estimate of the number of rows that are uploaded. You do this with the dbmlsync -urc option.

See "-urc dbmlsync option" [*MobiLink - Client Administration*].

Use background synchronization

From the remote user's point of view, the more synchronization happens in the background, the less urgent it is for synchronizations to be as fast as possible. Consider designing your remote application to use background synchronization so that remote users can continue to work even when synchronizing.

Key factors influencing MobiLink performance

The overall performance of any system, including throughput for MobiLink synchronization, is usually limited by a bottleneck at one point in the system. For MobiLink synchronization, the following might be the bottlenecks limiting synchronization throughput:

- **The performance of the consolidated database** Of particular importance for MobiLink is the speed at which the consolidated database can execute the MobiLink scripts. Multiple database worker threads can execute scripts simultaneously, so for best throughput you need to avoid database contention in your synchronization scripts.
- **The number of MobiLink database worker threads** A smaller number of threads involve fewer database connections, less chance of contention in the consolidated database and less operating system overhead. However, too small a number may leave clients waiting for a free database worker thread, or have fewer connections to the consolidated database than it can overlap efficiently.
- **The bandwidth for client-to-MobiLink communications** For slow connections, such as those over dial-up or wide-area wireless networks, the network may cause clients and MobiLink servers to wait for data to be transferred.
- **The client processing speed** Slow client processing speed is more likely to be a bottleneck in downloads than uploads, since downloads involve more client processing as rows and indexes are written.
- **The speed of the computer running the MobiLink server** If the processing power of the computer running MobiLink is slow, or if it does not have enough memory for the MobiLink database worker threads and buffers, then MobiLink execution speed could be a synchronization bottleneck. The MobiLink server's performance depends little on disk speed as long as the buffers and database worker threads fit in physical memory.
- **The bandwidth for MobiLink to consolidated communication** This is unlikely to be a bottleneck if both MobiLink and the consolidated database are running on the same computer, or if they are on separate computers connected by a high-speed network.

Tuning MobiLink for performance

The key to achieving optimal MobiLink synchronization throughput is to have multiple synchronizations occurring simultaneously and executing efficiently. To enable multiple simultaneous synchronizations, MobiLink uses pools of database worker threads for different tasks. One pool is dedicated to reading upload data from the network and unpacking it. Another pool of threads, called **database worker threads**, applies the upload to the consolidated database and fetches data to be downloaded from the consolidated database. Another pool of database worker threads is dedicated to packing and sending the download data to the remote databases. Each database worker thread uses a single connection to the consolidated database for applying and fetching changes, using your synchronization scripts.

Contention

The most important factor is to avoid database contention in your synchronization scripts. Just as with any other multi-client use of a database, you want to minimize database contention when clients are

simultaneously accessing a database. Database rows that must be modified by each synchronization can increase contention. For example, if your scripts increment a counter in a row, then updating that counter can be a bottleneck.

Synchronization requests are accepted (up to the limit specified by the -sm option) and the uploaded data is read and unpacked so that it is ready for a database worker thread. If there are more synchronizations than database worker threads, the excess are queued, waiting for a free database worker thread.

You can control the number of database worker threads and connections, but MobiLink always ensures that there is at least one connection per database worker thread. If there are more connections than database worker threads, the excess connections are idle. Excess connections may be useful with multiple script versions, as discussed below.

Number of database worker threads

Other than contention in your synchronization scripts, the most important factor for synchronization throughput is the number of database worker threads. The number of database worker threads controls how many synchronizations can proceed simultaneously in the consolidated database.

Testing is vital to determine the optimum number of database worker threads.

Increasing the number of database worker threads allows more overlapping synchronizations to access the consolidated database, and **may** increase throughput. However, it also increases resource and database contention between the overlapping synchronizations, and potentially increases the time for individual synchronizations. As the number of database worker threads is increased, the benefit of more simultaneous synchronizations becomes outweighed by the cost of longer individual synchronizations, and adding more database worker threads decreases throughput. Experimentation is required to determine the optimal number of database worker threads for your situation, but the following may help to guide you.

For uploads, performance testing shows that the best throughput can typically be achieved with three to ten database worker threads. Variation depends on factors like the type of consolidated database, data volume, database schema, the complexity of the synchronization scripts, and the hardware used. The bottleneck is usually due to contention between database worker threads executing the SQL of your upload scripts at the same time in the consolidated database.

When download acknowledgements are not used (the default), the client-to-MobiLink bandwidth is less influential because a database worker thread is free to process other synchronizations while other threads send the download. So, the number of database worker threads is less critical.

Many downloads can be sent concurrently—far more than the number of database worker threads. For optimal download performance, it is important for the MobiLink server to have enough RAM to buffer these downloads. Otherwise the download is paged to disk and download performance may degrade. To specify the MobiLink server memory cache size, use the -cmax option.

See “-cmax mlsrv12 option” on page 42.

If the MobiLink server starts paging to disk (possibly because of too many downloads being processed concurrently), consider using the -sm option to either decrease the number of database worker threads or limit the total number of synchronizations being actively processed.

See “-sm mlsrv12 option” on page 67.

Leaving download acknowledgement off (the default) can reduce the optimal number of database worker threads for download, because database worker threads do not have to process download acknowledgement transactions.

See "SendDownloadAck (sa) extended option" [*MobiLink - Client Administration*].

For download acknowledgement, blocking download acknowledgement has been discontinued, so all download acknowledgement is handled as non-blocking, which has better performance. With non-blocking acknowledgement, the server reuses the database worker thread while the remote database applies the download. This means that the number of database worker threads may not need to be increased, which results in better performance.

To get both the best download throughput and the best upload throughput, MobiLink provides two options. You can specify a total number of database worker threads to optimize downloads. You can also limit the number that can simultaneously apply uploads to optimize upload throughput.

The -w option controls the total number of database worker threads. The default is five.

The -wu option limits the number of database worker threads that can simultaneously apply uploads to the consolidated database. By default, all database worker threads can apply uploads simultaneously, but that can cause severe contention in the consolidated database. The -wu option lets you reduce that contention while still having a larger number of database worker threads to optimize the fetching of download data. The -wu option only has an effect if the number is less than the total number of database worker threads.

See "-w mlsrv12 option" on page 74 and "-wu mlsrv12 option" on page 75.

MobiLink database connections

MobiLink creates a database connection for each database worker thread. You can use the -cn option to specify that MobiLink create a larger pool of database connections, but any excess connections are idle unless MobiLink needs to close a connection or use a different script version.

There are two cases where MobiLink closes a database connection and open a new one. The first case is if an error occurs. The second case is if the client requests a synchronization script version, and none of the available connections have already used that synchronization script version.

Note
Each database connection is associated with a script version. To change the script version, the connection must be closed and reopened.

If you routinely use more than one script version, you can reduce the need for MobiLink to close and open connections by increasing the number of connections. You can eliminate the need completely if the number of connections used for synchronizations is the number of database worker threads times the number of script versions.

An example of tuning MobiLink for two script versions is given in the following command line:

```
mlsrv12 -c "dsn=SQL Anywhere 12 Demo" -w 5 -cn 10
```


Since the maximum number of database connections used for synchronizations is the number of script versions times the number of database worker threads, setting -cn to 10 ensures that database connections are not closed and opened excessively.

See "-cn mlsrv12 option" on page 43.

Monitoring MobiLink performance

There are a variety of tools available to help you monitor the performance of your synchronizations.

The MobiLink Monitor is a graphical tool for monitoring synchronizations. It allows you to see the time taken by every aspect of the synchronization.

See "MobiLink Monitor" on page 154.

In addition, there are several MobiLink scripts that are available for monitoring synchronizations. These scripts allow you to use performance statistics in your business logic. You may, for example, want to store the performance information for future analysis, or alert a DBA if a synchronization takes too long. You must write these scripts with the same care as your other scripts, avoiding contention and blocking as much as possible. For more information, see:

- "download_statistics connection event" on page 351
- "download_statistics table event" on page 354
- "synchronization_statistics connection event" on page 424
- "synchronization_statistics table event" on page 427
- "time_statistics connection event" on page 430
- "time_statistics table event" on page 433
- "upload_statistics connection event" on page 509
- "upload_statistics table event" on page 514

Central administration of remote databases

SQL Anywhere allows a central administrator to manage remote databases involved in MobiLink synchronization.

Central administration of remote databases lets you do the following:

- Centrally control when a remote database synchronizes with MobiLink.
- Perform schema changes on remote databases.
- Diagnose problems with specific remote databases or with the synchronization system in general.
- Upload log files.

Central administration of remote databases replaces the SQL passthrough functionality, originally introduced in version 11.0.0, that allowed you to download scripts of SQL statements from a consolidated

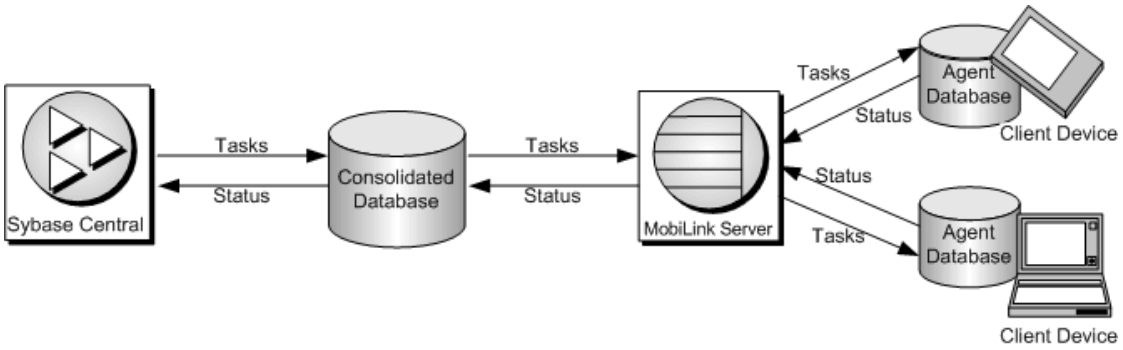
database to a SQL Anywhere or UltraLite client, and have those SQL statements executed on the client at an appropriate time.

Central administration is achieved using remote tasks. A remote task is an ordered collection of commands (similar to a batch file) that you create using the MobiLink 12 plug-in for Sybase Central. A remote task may have conditions that govern when it runs. A remote task can be configured to run only once, to run on demand, or to repeat regularly. Remote tasks are deployed to the consolidated database and can be assigned to one or more MobiLink Agents.

The MobiLink Agent is an application that runs on a remote device. Each set of databases being centrally administered from a specific MobiLink server must have one instance of the MobiLink Agent running. The Agent's job is to execute the remote tasks that are assigned to it at appropriate times. The MobiLink Agent uses an UltraLite database, which is called the agent database, to store tasks that have been assigned to it, the results of tasks that have been run and other configuration information.

Periodically, the MobiLink Agent synchronizes its agent database using regular MobiLink synchronization. During these synchronizations, the Agent receives any new tasks that have been assigned to it and optionally uploads results of tasks it has run so that they can be reviewed by the system administrator using the MobiLink 12 plug-in for Sybase Central.

The diagram below shows how information flows between Sybase Central, the consolidated database, and the client devices when using the central administration of remote databases feature.



Central administration concepts

The following concepts are important to understand when working with central administration of remote databases.

MobiLink project

A MobiLink project is created by an administrator using the MobiLink 12 plug-in for Sybase Central. You must first define a MobiLink project before you can work with remote tasks.

A MobiLink project is a collection of the following:

- Zero or more remote tasks.

- At least one connection to a consolidated database.
- Zero or more synchronization models.

MobiLink Agent

A MobiLink Agent is an application that runs on the client device. The Agent's purpose is to receive and execute tasks from the MobiLink server and report the status of those tasks back to the MobiLink server.

The MobiLink Agent can manage multiple remote databases on the client device. If remote databases on the client device need to synchronize with different consolidated databases, the device would require a different MobiLink Agent for each different consolidated database with which the application needs to synchronize.

MobiLink Agent ID

The MobiLink Agent ID is a string that identifies an Agent running on a client device to the MobiLink server. It can be viewed by the administrator working with the MobiLink project. Since each Agent runs on a single client device, this ID also identifies the device to the administrator.

Each MobiLink Agent must have a unique ID. The Agent ID can either be specified at start-up with the `mlagent` command or a default value is assigned in the form `Agent_`*`computername_UUID`*, where *computername* is the host name of the machine that the Agent is running on and *UUID* is a universally unique identifier.

Remote task

A unit of work is called a remote task. A remote task is a collection of commands. The MobiLink Agent receives work to do in the form of remote tasks, and reports the status of work it has attempted back to the administrator.

Command

A command is an instruction in a task that carries out some action. A task can have several commands and there is a set order to the commands. A command includes an action to perform, input parameters, and instructions about what to do if the command fails.

Deployed task

A deployed task is a task that has been copied into the consolidated database. Only deployed tasks can be assigned to an Agent for execution.

Status information

Status information is information about remote tasks, such as whether or not the tasks completed successfully. This information is stored on the client in the agent database when tasks execute, and sent to the server at various times so the administrator can see the status of the remote tasks in the system.

Agent database

An Agent database is an UltraLite database on the remote device that is used by the MobiLink Agent to store information about tasks and configuration.

The default location of the Agent database is *%ALLUSERSPROFILE%\Application Data\SQL Anywhere 12\MobiLink Agent* on Windows and *My Device\Application Data\SQLAny12\MLAgent* on Windows Mobile.

The Agent database file name is whatever was specified with -n option for *mlagent.exe*, plus the extension *.udb*. If the -n option is not provided, the default name is *mlagent.udb*.

Remote database

A remote database is an UltraLite or SQL Anywhere database on a remote device that contains your application data, is involved in MobiLink synchronization, and is managed by a MobiLink Agent. Each remote database has a remote schema name that identifies its schema.

Remote schema name

A remote schema name identifies a group of databases with the same schema. Typically all databases with the same remote schema name will be databases for the same version of an application. A schema includes things like: table definitions, stored procedures, triggers, publications and synchronization profiles. A schema does not include items that would normally vary from one instance of a database to another such as synchronization users and database users.

A remote database cannot be managed remotely unless it has a remote schema name, so at least one remote schema name must be defined before an Agent can be created in Sybase Central.

Server-initiated remote task (SIRT)

A server-initiated remote task is any remote task that is run when the Agent receives notification from the server to run the task. A task may have a schedule, but still be initiated by the server.

Setup overview

The basic steps to set up central administration on the server and client are as follows.

To set up central administration on the server

1. Create a MobiLink project in Sybase Central. See "Creating a MobiLink project" [*MobiLink - Getting Started*].
2. Use Sybase Central to define one or more remote schema names to identify your remote database(s) to the system.
3. Use Sybase Central to make all the Agents that are managing the remote databases known to your system.
4. Use Sybase Central to create tasks and assign them to Agents. See "Working with remote tasks" on page 139.

To set up central administration on the client

1. Configure the MobiLink Agent on each device running the application. This gives it an identity in the system and provides the MobiLink connection information.
2. Run the MobiLink Agent on the device so it is able to receive tasks from the server and execute them. See "Configuring and running the MobiLink Agent on the client device" on page 134.

Working with MobiLink Agents

The Agent manages the execution of remote tasks on a device. It stores tasks to be executed and the results of tasks it has executed in the agent database. The Agent synchronizes the agent database using ordinary MobiLink synchronization. During synchronization, the Agent receives new tasks to execute and uploads information about tasks it has executed.

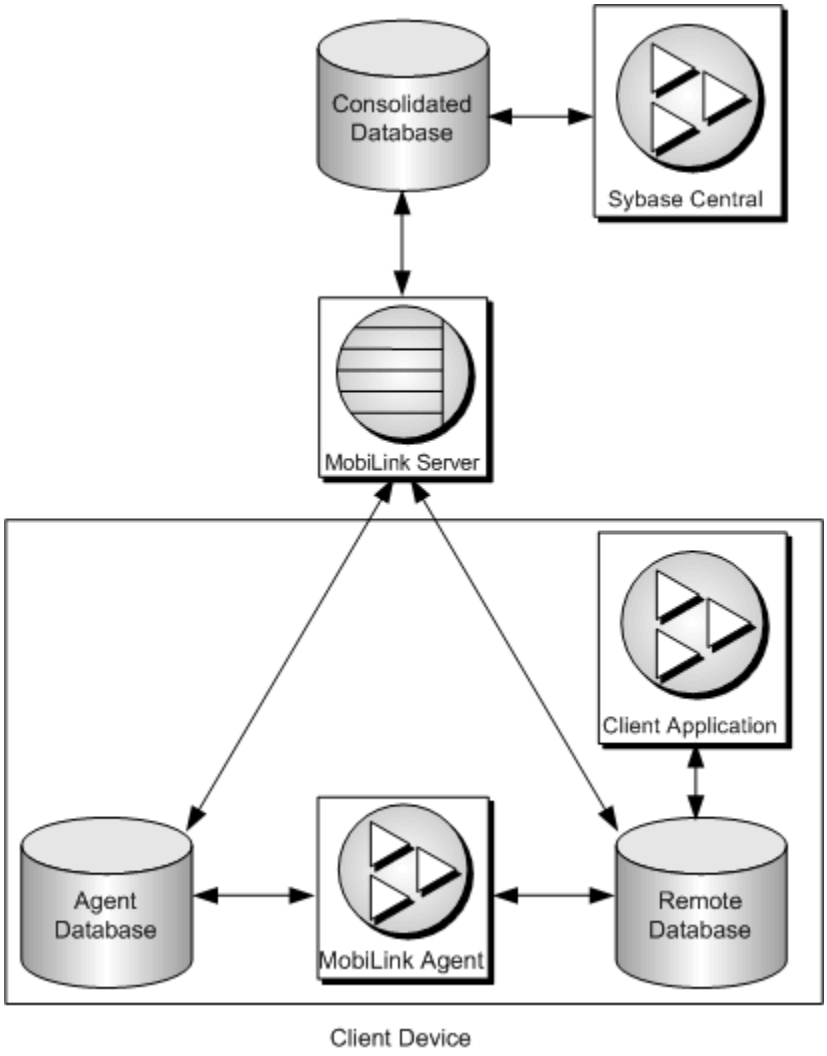
The Agent synchronizes the agent database when the following occur:

- The Agent is started.
- The Agent receives a notification from the server to synchronize the agent database. The Agent listens for notifications from the server and when notified, synchronizes the agent database.
- A user-specified amount of time has elapsed since the last synchronization.
- A task completes that is configured to send its status immediately upon completion.

The Agent is multi-threaded and may run more than one task in parallel. A task is only deleted after it enters a "final" state. Final states are: successful, failed, expired or canceled. A task only enters a final state if it is "run exclusive" or "run immediate" and it succeeds or fails; or if it is a scheduled task and it expires; or it is canceled at the server. The "on demand" tasks do not enter a final state unless they are canceled. On demand tasks sit in the Agent and are executed by server-initiated requests (SIRT).

The MobiLink Agent is not the same as the QAnywhere Agent and the two do not conflict with one another.

The following diagram shows the flow of communication to and from the Agent.



Configuring and running the MobiLink Agent on the client device

The Agent can be run in two modes: configuration mode and normal mode. In configuration mode, options specified on the command line are stored in the agent database for use during the next run in normal mode. Once the specified options are stored, the Agent terminates.

When run in normal mode, the Agent reads the configuration options stored in the agent database and remains running. While running it executes remote tasks it has received when appropriate and synchronizes the agent database at various times to receive new remote tasks and to upload results of remote tasks that it has run.

When run in normal mode, the Agent always attempts to do a synchronization at startup. This can be useful when you want to force the Agent to get up to date information from MobiLink.

Syntax

mlagent [*options*]

To run in configuration mode, specify -c or -cr on the mlagent command line.

Option	Description
@*data*	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used. See "Using configuration files" [*SQL Anywhere Server - Database Administration*].
-c	Set the configuration options, only updating those options that differ from the current options, then stop the Agent.
-cr	Set the configuration options, resetting all existing options to the defaults, then stop the Agent. Using this option resets all the information in the agent database and should only be used if the agent database is in an unrecoverable state.
-a *agentid*	Valid only with -c or -cr. Specify the ID of this Agent. If the -a option is not specified, the default is `Agent_computername_UUID`, where *computername* is the host name of the machine that the Agent is running on and *UUID* is a universally unique identifier.
-db *database location*	Valid only with -c or -cr. Specify the path where remote databases are stored on the client device.
-x *protocol*[*protocol-options*] ... *protocol* : **tcpip** | **tls** | **http** | **https** *protocol-options*: (*option=value*; ...)	Valid only with -c or -cr. Specify the MobiLink stream parameters. These parameters determine how the Agent connects to the MobiLink server when synchronizing the agent database.
-ap *parameters*	Specify the MobiLink authentication parameters used when synchronizing the agent database.
-ek *key*	Specify the encryption key used to access the agent database.
-n *name*	Specify the name of the agent database. The default is taskdb.
-o *file*	Log output to the specified file.
-on *size*	Append *.old* to the log file name and start a new file with the original name when the log reaches the *size* specified. Size must be a minimum of 10K. This option cannot be used with -os.

Option	Description
-os *size*	Rename the log file to *YYMMDDxx.mla* and start a new file with the original name when the log reaches the *size* specified. Size must be a minimum of 10K. This option cannot be used with -on.
-ot *file*	Truncate the file and log output messages to it.
-p *password*	Specify the MobiLink password used to synchronize the agent database.
-pi	Causes the Agent to do a ping synchronization of its Agent database with the MobiLink server, using the currently configured MobiLink client network protocol options and user authentication parameters. The mlagent process immediately returns 0 if the ping synchronization was successful, or the SQL error code of the synchronization request if the synchronization failed. The MobiLink Agent database must be configured by running mlagent with -c or -cr before mlagent is invoked with -pi. The MobiLink Agent cannot be invoked with both -pi and either -c or -cr on the command line.
-q	Run in a minimized window.
-qi	Do not display tray icon or window.
-u *user*	Specify the MobiLink user name used to synchronize the agent database.
-v*level*	Specify the output verbosity level from 0-9. The default is 3.

Example

The following example demonstrates how to run the Agent in configuration mode. It uses the default Agent database and sets the agent ID to be the same as the MobiLink user ID:

```
mlagent -c -a username -u username -p password -x
http{host=myhost.example.com;port=8080} -o logfile.mla
```

The following example demonstrates how to run the Agent in normal mode, accepting all the settings from the default agent database:

```
mlagent
```

Stopping the MobiLink Agent

The MobiLink Agent stop utility lets you stop an instance of the MobiLink Agent running on the same device where the stop utility is run.

Syntax

mlastop [*options*]

Option	Description
-n *name*	The name of the Agent to stop. If -n is not specified, all Agents on the device are stopped.

For information about deploying the Agent, see "Deploying SQL Anywhere MobiLink clients" on page 793 and "Deploying UltraLite MobiLink clients" on page 795.

Working with Agents in Sybase Central

After an Agent is created and configured on the client device using the mlagent command, the Agent must also be created in Sybase Central before it can be assigned tasks. The **Create MobiLink Agent Wizard** guides you through the creation of the Agent, where you specify the information that is required for the Agent to be properly identified.

Note that you must have a remote schema name defined before you can create an Agent in Sybase Central.

To add a remote schema name (Sybase Central)

1. Ensure the MobiLink project you are working with is selected in the left pane in **Folders** view, right-click **Remote Schema Names** and choose **New** » **Remote Schema Name**.
2. Follow the instructions in the **Create Remote Schema Name Wizard** and click **Finish**.

To add an Agent (Sybase Central)

1. Ensure the MobiLink project you are working with is selected in the left pane in **Folders** view, expand **Consolidated Databases**, right-click **Agents** and choose **New** » **Agent**.
2. Follow the steps in the **Create MobiLink Agent Wizard**.

Once the Agent is created in Sybase Central, you can do the following:

- view and change Agent properties
- add remote databases for an Agent to manage
- synchronize an Agent
- delete an Agent
- view events for an Agent
- view tasks for an Agent
- view information about remote databases managed by an Agent

Viewing and changing Agent properties

Agent properties can be viewed and edited from the **Agent Properties** window in Sybase Central. You can also set the following properties by right-clicking an Agent and choosing **Set**, then the property you want to set:

- **Synchronization Interval**
- **Administration Polling Interval**
- **MobiLink Client Network Protocol Options**
- **Connection Strings for Managed Databases**

The MobiLink client network protocol options are an Agent property that is specified on the client and sent up to the server when the Agent first synchronizes. If an administrator changes an agent's MobiLink protocol options, the new value is sent to the Agent when it synchronizes, and the Agent uses the new value for all subsequent communicates with MobiLink.

If an administrator sends invalid MobiLink communication options (for example, an incorrectly set server host name) it may become impossible for the Agent receiving that value to communicate with the MobiLink server any more. In this case, the administrator must correct the MobiLink client network protocol option in Sybase Central, and then have the Agent re-configured on the device to the correct set of options.

To view or change Agent properties (Sybase Central)

1. Ensure the consolidate database you are working with is expanded in the **Folders** view of the MobiLink 12 plug-in for Sybase Central. Expand **Agents**, right-click the Agent you want to worth with and choose **Properties**.
2. If necessary, make changes to the properties and click **Apply**.

Adding managed remote databases

In order for an Agent to manage a remote database, the database must be associated with the Agent in Sybase Central. This is done by specifying a remote schema name.

To add a managed remote database (Sybase Central)

1. Ensure the consolidate database you are working with is expanded in the **Folders** view of the MobiLink 12 plug-in for Sybase Central. Expand **Agents**, right-click the Agent you want to worth with and choose **Add Managed Remote Database**.
2. The available remote databases are represented by remote schema names. Choose a **Remote Schema Name** from the dropdown list.
3. Enter a connection string for the remote database and click **OK**.

   The connection string is used on the client device. All ODBC data sources or paths and/or files must be valid for the device.

Adding a group

A group is a collection of MobiLink Agents. You must have one of more Agents defined before you can create a group.

To add a group (Sybase Central)

1. Ensure that your consolidated databases are running.
2. Ensure the MobiLink project you are working with is selected in the **Folders** view in the left pane, then right-click on the project name and choose **New** » **Group**.
3. Follow the instructions in the **Group Wizard** and click **Finish**.

Agent authentication

The Agent acts as a MobiLink synchronization client when synchronizing the agent database. The synchronization scripts to support the Agent use the *ml_ra_agent_12* script version. The Agent synchronization scripts are automatically installed when you do a MobiLink setup for a consolidated database. However, no authentication scripts are provided for the Agent. See "Setting up a consolidated database" on page 2.

At least one of the following must be defined in order to have a secure system:

- "authenticate_user connection event" on page 315
- "authenticate_user_hashed connection event" on page 320
- "authenticate_parameters connection event" on page 312

This can be done one of the following ways:

- Call the ml_add_connection_script stored procedure. For example,

  ```
  ml_add_connection_script( 'ml_ra_agent_12', 'authenticate_user', '<DEFINE
  YOUR AUTHENTICATION LOGIC>' )
  ```

- Use **Connection Scripts** in Sybase Central. Note that you can use the ml_ra_agent_12 script version or ml_global. It is likely that you will want to have the same authentication for both application data synchronization and for the Agent. By using ml_global, you can just define one set of authentication scripts for both. This is the recommended way to do authentication. See "ml_global script version" on page 283.

Working with remote tasks

A remote task is the unit of work when performing central administration of remote databases. A remote task consists of the following:

- One or more trigger mechanisms
- Optional conditions
- An ordered collection of commands
- Other properties

Tasks are created by the administrator using the MobiLink 12 plug-in for Sybase Central. At design-time they are stored locally on the administrator's computer in the project. For a list of commands that can be used to build tasks, see "Commands" on page 144.

Generally, one task should not depend on the completion of another task. However, it is possible to create a task that depends on another by having one task write something to the remote database and having the condition of another task query that value.

Once the administrator is ready for Agents to receive a task, the administrator deploys the task. When deployed, a task is copied into the consolidated database. There are now two copies of the task: the deployed task in the consolidated database and the design-time task in the project. Deployed tasks may not be modified. However, the design-time task used to create them may be modified and deployed again (to create a second deployed task). Deployed tasks can be cancelled, initiated (SIRT), reactivated and assigned to new recipents.

Once deployed, a task may be assigned to one or more Agents for execution. When assigned to an Agent, the task is downloaded to the Agent. The Agent then executes the task at an appropriate time and optionally uploads the results back to the consolidated database where they can be reviewed by an administrator using the MobiLink 12 plug-in for Sybase Central.

Tasks have the following attributes:

- **Name** A remote task has two names: one identifies the design-time version of the task stored in the project while the other identifies the task once it is deployed. Often the two names are the same.

  A task's design-time name is assigned when the task is created and must be unique among tasks in the project. A deployed task name is assigned when the task is deployed and this name must be unique among deployed tasks in the consolidated database.

- **Description** A description of the task can be entered and may contain any text you wish to associate with the task. The description is stored in the project and in the consolidated database (once the task is deployed), but is not sent to the Agent.

- **Trigger mechanisms** A remote task's trigger mechanisms determine when the Agent attempts to execute the task. A task may have more than one trigger mechanism. There are three supported trigger mechanisms:

  - **Based on a schedule** The task is triggered at specific times or at specific time intervals. This option must be explicitly set for a task.

  - **When received by an Agent** The task is triggered when it is received by the Agent, and will run only once. This option must be explicitly set for a task.

    - **On demand** The task may be triggered at any time by a message from the server in a process called Server-Initiated Remote Tasks (SIRT). All tasks that are not configured to run only once support being triggered **on demand**.

- **Conditions** Before a task can be executed, all its conditions must be satisfied. If a task is triggered but all its conditions are not met, then the run attempt is considered a failure and the task must be triggered again before the conditions are reevaluated.

- **Remote schema name** A task can optionally be associated with a remote schema name. Tasks that require access to a remote database must be associated with a remote schema name. The remote schema name is used by the Agent to determine which of the databases the Agent needs to access when executing the task. Tasks must be associated with a remote schema name if they contain any of the following commands: create database, drop database, execute SQL or synchronize. A remote task must also be associated with a remote schema name if you want to use a SQL condition to determine if the task can run.

- **Commands** A task contains an ordered set of commands that carry out the work required for the task. The order in which the commands are specified defines the order in which the commands are executed within the task. It is important to be aware of the order of the commands because commands could be dependent on each other. See "Commands" on page 144.

- **Maximum number of retries** Each command has an **on failure** action that can be used to cause the task or the command to be retried if the command fails. This option lets you limit the number of retries allowed during a single run attempt.

- **Delay between retries** This option specifies the amount of time to wait after a command fails before attempting to retry the command or task. This delay may allow a transitory condition (such as a locked database table or a locked file) that caused the failure to pass before the command or task is reattempted. Retry delays are assumed to be "short" periods of time. A task condition is not re-evaluated between retries.

- **Maximum running time** It is possible that a task when executed, does not behave as the administrator intended. An OS call could hang; an attempt to synchronize could be very slow—a SQL statement could be blocked on another connection using the database. Setting a maximum running time for a task lets you limit how long the task may run. If the maximum running time is reached, the task is terminated (the actual time at which the task is terminated depends on the ability to interrupt the operation). The status for the task is set to reflect the timeout and the task is not retried until it is triggered again. If a command in a task fails and the task or command needs to be retried, the maximum running time is reset after the delay. So, the maximum running time is considered to be per attempted task execution or retry. It does not include the aggregate time of all retries and it does not include Prompt commands.

- **Schema change** Schema change tasks change the schema of a remote database. If the task succeeds, the remote schema name of the remote database is also updated. Schema change tasks are always high priority tasks and they report their status on completion.

- **High priority** High priority tasks are always triggered when received by an Agent and executed as soon as any currently executing tasks are complete. They may not be executed based on a schedule and they may not have any conditions on their execution. A task marked high priority is only executed

when there are no other tasks being executed to ensure that no other task interferes with the execution of a high priority task.

- **Status reporting** Status reporting options on a task allow you to specify when and if results are reported back to the consolidated database, both when the task completes successfully and when it fails. The available options are:
  - **Send status only** Task results are not reported. However, information about the number of task successes or failures is maintained and reported.
  - **Return results the next time the Agent synchronizes the agent database** Task results and status are maintained and reported the next time the Agent synchronizes the agent database.
  - **Return results immediately when task execution completes** Task results and status are reported as soon as the task completes.

Remote task logic

The following is an outline of the logic used to execute a remote task.

```
current_command = 1;
num_tries = 0;
EXECUTE_TASK:
loop {
      num_tries = num_tries + 1;
      EXECUTE_COMMANDS;
      if( task_success or task_abort ) break EXECUTE_TASK;
      if( task_retry and at maximum tries ) {
            break EXECUTE_TASK;
      } else {
            continue;
      }
}

EXECUTE_COMMANDS:
for each command starting at current_command {
      execute current_command;
      if( command failed ) {
            if( action on failure is "abort task" ) {
                  break EXECUTE_COMMANDS, returning task_abort;
            } else if( action on failure is "continue" ) {
                  // no action, continue at next command
            } else if( action on failure is "retry task" ) {
                  current_command = 1;
                  break EXECUTE_COMMANDS, returning task_retry;
            } else if( action on failure is "retry command" ) {
                  // no change to current_command
                  break EXECUTE_COMMANDS, returning task_retry;
            }
      }
      current_command = next command number;
}
return task_success;
```

The maximum running time is reset for a task when the task starts and when a command or the entire task is retried because a command fails.

Notice that when a command or task is retried because a command fails, the condition is not reevaluated and no new trigger event is required. It is important to consider whether properties referenced in the

condition can change during the execution of the task, and whether retrying a command if the condition fails would produced undesirable results.

Creating a remote task

Tasks are defined and managed within the context of a MobiLink project. An administrator can create, alter and remove a task without requiring a connection to the consolidated database. Tasks that have not been deployed are considered to be in development and exist locally in the MobiLink project only.

Follow the procedure below to create a new task. You create the task first, and then it can be deployed to the consolidated, and then assigned to recipients (Agents):

To create a task (Sybase Central)

1. Ensure the MobiLink project you are working with is selected in the left pane in **Folders** view, then right-click **Remote Tasks** and choose **New** » **Remote Task**.
2. Follow the steps in the **Create Remote Task Wizard** and click **Finish**.

   Note
   If you need to create a new remote schema name, which identifies a group of remote databases that have the same schema, click **Create a Remote Schema Name** on the **Welcome** page of the **Create Remote Task Wizard**. The **Create Remote Schema Name Wizard** appears and the **Create Remote Task Wizard** remains open. Once you have created the new remote schema name, it appears in the **Remote Schema Name** dropdown of the **Create Remote Task Wizard**.
3. Follow the procedure to add one or more commands to the task. See "Adding a command" on page 149.

Editing a remote task

Follow the procedure below to make changes to a previously defined remote task.

To edit a remote task (Sybase Central)

1. Ensure the MobiLink project you are working with is selected in the **Folders** view in the left pane, then right-click the remote task you want to edit and choose **Properties**.
2. Edit the remote task properties.
3. When you are finished editing the remote task properties, click **Apply**.

Deploying a remote task

When you are ready to add the task to the system, the task needs to be deployed. Deploying a remote task means that the task is copied into the consolidated database and the new copy is given a name. The deployed task name is often the same name that the task had during development.

When you deploy a task you can also assign it to a list of Agents. Agents can be added after a task is deployed, so it is not necessary to add them when the task is first deployed. This is useful when new Agents are added to the system after a task has already been deployed.

A remote task must contain at least one command before it can be deployed. For information about commands, see "Commands" on page 144 and "Adding a command" on page 149.

To deploy a remote task (Sybase Central)

1. Ensure the MobiLink project you are working with is selected in the **Folders** view in the left pane, then right-click the remote task you want to deploy and choose **Deploy**.
2. Follow the instructions in the **Deploy a Remote Task Wizard**.
3. Click **Finish**.

Commands

A command is an instruction in a task that carries out some action. A task can have several commands and there is a set order to the commands. A command includes an action to perform, input parameters, and instructions about what to do if the command fails.

Copy file command

Makes a copy of a file on the remote device.

Parameters

- **Original file name** The name of the file to be copied.
- **New file name** The name of the file to which the original file is to be copied.
- **Overwrite existing file if necessary** Using this option causes the file to be copied even if a file with the name specified with the **New file name** parameter already exists.
- **Ignore read-only attribute** This parameter can only be used if the **Overwrite existing file...** parameter is used. When this option is used, the copy occurs even if a file with the name specified with the **New file name** parameter already exists, and is read-only.

Remarks

File name specifications can be absolute or relative. If a relative file name is specified, it is taken to be relative to the current working directory of the Agent.

Create database command

Creates a new remote database on the remote device that is managed by the Agent.

Parameters

- **File name** The file name for the new database. Usually you should use the {db_location} macro to specify the path for your database. For example, *{db_location}\mydatabase.db*. If an Agent is going to manage more than one SQL Anywhere database then you must create each database in a separate directory. It is recommended that you place each database in a separate subdirectory of the *{db_location}* directory.

- **CHAR collation** Specifies the collation for CHAR, VARCHAR and LONG VARCHAR data types in the new database.

- **NCHAR collation** For SQL Anywhere only, specifies the collation for NCHAR, NVARCHAR and LONG NVARCHAR data types in the new database.

Remarks

This command can only be used in a remote task marked as **Requiring or Creating a Remote Database**.

The type of database (UltraLite or SQL Anywhere) created by the command is determined by the remote schema name specified for the remote task. If the file name specified uses a directory that does not exist on the remote device then the directory is created. The file name specification may be absolute or relative. If a relative file name is specified, it is taken to be relative to the current working directory of the Agent. This is not supported for SQL Anywhere in Windows Mobile.

The CREATE DATABASE statement can be used to initialize a database on a desktop computer, which can later be copied to a Windows Mobile device.

Delete file command

Deletes a file on the remote device.

Parameters

- **File name** The name of the file on the remote database to be deleted.

- **Ignore read-only attribute** Checking this option results in the file being deleted even if it is marked read-only.

Remarks

The file name specification may be absolute or relative. If a relative file name is specified, it is taken to be relative to the current working directory of the Agent.

Download file command

Download a file from the server to the remote device.

Parameters

- **Server file name** The name of the file on the server to download to the remote device. The file name can not be absolute. It is taken relative to the download root directory of the MobiLink server. This directory is specified using the -ftr option on the MobiLink server. See "-ftr mlsrv12 option" on page 50.

- **Remote file name** Specifies the location on the remote device where the file is to be stored. The file name may be absolute or relative. If a relative file name is specified, it is taken to be relative to the current working directory of the Agent.

Drop database command

Deletes a managed remote database.

Parameters

None

Remarks

This command can only be used in a remote task marked as requiring or creating a remote database. The database dropped is the one associated with the remote schema name specified for the remote task.

The connection string for the remote schema name associated with the task that contains the drop database command must have a DBF parameter.

The database being dropped must not be running when the drop database command is executed.

All data in the remote database is lost when the database is dropped.

Dropping a database is not supported on Windows Mobile.

See also

- "UltraLite DBF connection parameter" [*UltraLite - Database Management and Reference*]
- "DatabaseFile (DBF) connection parameter" [*SQL Anywhere Server - Database Administration*]

Execute SQL

Executes SQL against a remote database.

Parameters

- **SQL** The SQL to be executed. Separate SQL statements with GO on a line by itself. For SQL statements bracketed by a BEGIN and END statement, do not specify GO within the BEGIN-END block. Here is an example of the correct use of GO to delimit statements:

```
SELECT * FROM systable
GO
```



```
CREATE PROCEDURE p1()
BEGIN
    CREATE TABLE t1( pk INTEGER PRIMARY KEY );
    INSERT INTO t1 VALUES( 5 );
    COMMIT;
END
GO
SELECT * FROM sysprocedure
```

Remarks

This command can only be used in a remote task marked as requiring or creating a remote database. The SQL is executed against the database associated with the remote schema name specified for the remote task.

When executing SQL, the Agent does not COMMIT any statements. If the SQL being execute does not have a COMMIT, the statement is rolled back. This is important when the SQL is INSERT, UPDATE and DELETE statements or any other statements that do not explicitly cause a COMMIT.

The status /results for the command store the results of the executed SQL. DDL statements return no results. INSERT/UPDATE/DELETE statements return the number of rows affected as a single value on a line. SELECT statements return the results in *.csv* format with column headings as the first row. Results from multiple statements are all appended into one big result.

Prompt command

Displays a message box on the remote device.

Parameters

- **Message** The message to be displayed in the message box.

Remarks

The message on the remote device is visible until it is dismissed by clicking **OK**.

If the task has additional commands that follow the prompt command, they are not executed until the message is dismissed. The time between when the prompt is displayed and when the user clicks **OK** is not included in the calculation of the task running time.

Rename file command

Renames a file on the remote device.

Parameters

- **Original file name** The current name of the file to be renamed.
- **New file name** The name of the file, after it is renamed.

- **Overwrite existing file if necessary** Checking this option causes the file to be renamed even if a file with the new file name already exists.

- **Ignore read-only attribute** This option can only be selected if **Overwrite existing file...** is selected. When this option is selected, the rename occurs even if a file with the new file name already exists and is read-only.

Remarks

File name specifications can be absolute or relative. If a relative file name is specified, it is taken to be relative to the current working directory of the Agent.

Run program command

Run a program on the remote device.

Parameters

- **Command line** The command line to be executed.

Remarks

Execution of the task will not continue until the program completes execution. The command is considered successful if the exit code for the program that was run is 0.

Synchronize command

Synchronizes a remote database.

Parameters

- **Synchronization profile** Specifies a synchronization profile already defined in the remote database that contains the options to be used for the synchronization.

- **Extra options** Specifies additional options to be used for the synchronization. If an option is specified in both the extra options and the synchronization profile, then the setting from the extra options overrides the setting in the synchronization profile. This option may be left blank.

Remarks

This command can only be used in a remote task marked as requiring or creating a remote database. The database synchronized is the one associated with the remote schema name specified for the remote task.

Upload file command

Uploads a file from the remote device to the server.

Parameters

- **Remote file name** The name of the file on the remote device to upload to the server. This file name can be absolute or relative. If a relative file name is specified, it is taken to be relative to the current working directory of the Agent.

- **Server file name** Specifies the location on the server where the file is to be stored. This file name can not be absolute and may not contain more than one backslash (\ ). It is taken relative to the upload root directory of the MobiLink server. This directory is specified using the -ftru option on the MobiLink server.

  It is a good idea to use a macro in the sever file name to ensure that each agent that executes the command uploads its file to a different location on the server. Otherwise, you may have problems with agents over-writting each other's files. A good convention is to place the files from each agent in a separate directory where the name of the directory is the agent id. You can use the {agent_id} macro to achieve this. For example, if you want to upload a file called *myuploadfile.txt* you might set the destination file name to `{agent_id}\myuploadfile.txt`. See "Variables in parameters" on page 150.

Adding a command

A remote task cannot be deployed until it contains at least one command. Follow the procedure below to add commands to a remote task. The following list shows the ways you can add a command to a task:

- In the **Folders** view in the left pane, right-click the task and choose **Add Command**.
- In the **Folders** view in the left pane, select the task and click the **Add Command** toolbar button.
- When you create a task, you automatically get a command in the right pane. Press Tab to move from parameter to parameter. If you keep pressing Tab, a new command is automatically added to the task.
- Right-click on an existing command and choose **Add Command**.
- Right-click in the whitespace under the existing commands and choose **Add Command**.
- Select the task in the **Folders** view in the left pane. From the **File** menu choose **Add Command**.

To add a command to a remote task (Sybase Central)

1. Ensure the MobiLink project you are working with is selected in the **Folders** view in the left pane.
2. Expand **Remote Tasks**, right-click the remote task you want to work with and choose **Add Command**. The **Commands** pane appears in the right pane.
3. From the **Command Type** dropdown list, choose the type of command required. For information on the available commands, see "Commands" on page 144.
4. Fill in the appropriate parameters for the selected command.

5. From the **On failure** dropdown list, choose one of the following options to specify how to proceed if the command fails:
   - **Abort Task** The current attempt to execute the task is terminated and the attempt is marked as failed.
   - **Continue** The task will continue to execute by moving to the next command.
   - **Retry Command** The task is retried beginning at the failed command. If the maximum number of retry attempts for the task is reached, the command is not retried.
   - **Restart Task** The task is retried, starting at the first command. If the maximum number of attempts for the task is reached, the task is not retried.

Variables in parameters

The following macros provide access to information that may vary between remote devices. These values may be used in remote task conditions, in parameters for commands within remote tasks and in connection strings:

Variable	Replacement
{ml_username}	The MobiLink user name (uid) being used by the Agent to synchronize with the agent database.
{ml_password}	The MobiLink password (pwd) being used by the Agent to synchronize with the agent database.
{ml_stream}	The protocol parameters for connecting to the MobiLink server. For example, `HTTP{host=Sybase.com;port=9376}`
{remote_id}	The remote ID for the remote database associated with this task. This value is only meaningful for in remote tasks that are marked as **Requiring or Creating a Remote Database**.
{agent_id}	The Agent ID.
{db_location}	The Agent's remote database directory, as specified by the `mlagent -db` option.
{battery_level}	The battery level for the remote device. The range is zero to one hundred (0-100).
{is_on_ac_power}	Indicates whether or not the remote device is using an AC power source. 1 indicates the device is plugged in and 0 indicates the device is running on battery power.
{rows_to_upload}	For UltraLite databases only. The number of rows in the remote database that will be uploaded if a full synchronization is done.

Variable	Replacement
{agent_log}	The full path and file name for the Agent log file. This file can be uploaded from the device to help diagnose problems when necessary. If the Agent runs without a log file, this variable is an empty string.
{agent_db}	The full path and file name for the agent database file. This file can be uploaded from the device to help diagnose problems when necessary.

Status

When the Agent runs a task it stores status information about that execution in the agent database unless the task is marked to not report any status information. Status information in the agent database is uploaded to the server whenever the agent database is synchronized.

The status information is accessible using the MobiLink 12 plug-in for Sybase Central. To view status for a specific deployed task, select the task in the **Folders** view in the left pane and view either the **Recipients** or **Results** tabs in the right pane. To view the status of all tasks assigned to a specific Agent, select the Agent in the **Folders** view in the left pane and view the **Tasks** tab in the right pane.

The following stored procedures also return status information:

- “ml_ra_get_agent_events system procedure” on page 718
- “ml_ra_get_agent_ids system procedure” on page 720
- “ml_ra_get_agent_properties system procedure” on page 721
- “ml_ra_get_latest_event_id system procedure” on page 722
- “ml_ra_get_orphan_taskdbs system procedure” on page 722
- “ml_ra_get_remote_ids system procedure” on page 723
- “ml_ra_get_task_results system procedure” on page 723
- “ml_ra_get_task_status system procedure” on page 725

System procedures

In addition to the functionality in Sybase Central for managing remote tasks, there are also MobiLink system procedures in the consolidated database that can be used to automate administration tasks. With the exception of the ml_ra_cancel_notification system procedure and the repair procedures, everything that can be done with system procedures can also be done using the MobiLink 12 plug-in for Sybase Central. However, the following tasks can only be done using the MobiLink 12 plug-in for Sybase Central:

- Create new tasks
- Create a new remote schema name
- Add descriptions to Agents, remotes, remote schema names and tasks

All the new MobiLink system tables, system procedures and the Agent script version begin with the prefix **ml_ra_**.

Following is a list of the system procedures used for central administration of remote tasks:

Deployment and configuration

The following section provide information about deployment and configuration. For a list of files required to deploy the MobiLink Agent, see "Deploying SQL Anywhere MobiLink clients" on page 793 and "Deploying UltraLite MobiLink clients" on page 795

Agent deployment considerations

In order to properly administer the remote databases in your MobiLink synchronization system, there are a few technical points to consider:

- The MobiLink Agent only runs on Windows and Windows Mobile devices. Remote databases on platforms other than these currently cannot be managed via a MobiLink Agent.

- Managing UltraLite remote databases requires using the UltraLite engine. This means that applications that accessing those remote databases must also do so via the UltraLite engine. Attempting to use the in-process version of UltraLite will result in file-in-use errors. See "Choosing an UltraLite data management component" [*UltraLite - Database Management and Reference*].

- Central administration is only possible when the MobiLink Agent is running on a device. In general, it is assumed that the Agent is always running on a device. You can stop an Agent using *mlastop.exe*;

however, the Agent will need to be re-started in order for central administration to be effective again. See "Stopping the MobiLink Agent" on page 136.

Deploying UltraLite applications and databases with the MobiLink Agent on Windows Mobile

There are various mechanisms to deploy UltraLite with the Agent on a Windows Mobile device.

You can use the SQL Anywhere **Deployment Wizard** to build a *.CAB* cabinet file that can be used to deploy SQL Anywhere on a Windows Mobile device. However, the **Deployment Wizard** does not include support for creating deployments of user applications and databases.

On completion of the **Deployment Wizard** an *.INF* file is created. The *.INF* file consists of a number of sections that describe the target location of the files, shortcuts, and registry settings contained within the *.CAB* file. This *.INF* file can be modified to include logic to install user applications and databases with the Agent.

Deploying with the SQL Anywhere for Windows Mobile Deployment Wizard

You can use the **SQL Anywhere for Windows Mobile Deployment Wizard** to deploy the files required for central administration of remote tasks.

The **SQL Anywhere for Windows Mobile Deployment Wizard** includes the option to **Manage SQL Anywhere Remote Databases** or **Manage UltraLite Remote Databases**.

Configuring the Agent (Windows desktop)

One way to configure the Agent on a Windows machine is to have an install program (the same one that installs your application) install the agent and run the commands to configure, validate and start the Agent. An install program could prompt for the MobiLink identification and/or authentication parameters. Those parameters could then be used to configure the agent using `mlagent -cr` on the command line.

After the agent is configured, the install program could run **mlagent** *-pi* ... to verify that authentication parameters are valid. A connection to the MobiLink server is required when this command is run to properly do authentication validation.

Lastly, the Agent could be launched as the final step of the install process. The Agent will then be ready to receive and execute tasks on the target device. The Agent's process return code can be used to provide information on the Agent's execution. For example, if the mlagent failed to ping the MobiLink server, the return code from *mlagent.exe* would be the SQLCODE that results from a synchronization of the Agent database with the configured mlagent options.

Configuring the Agent (Windows Mobile)

How you configure the MobiLink Agent on Windows Mobile depends on how the user's application will be installed on the device.

One way to configure and run the MobiLink Agent is to have the user application prompt for MobiLink identification and/or authentication parameters and then configure and/or launch the MobiLink Agent as part of its start up process.

- The application can attempt to run the Agent. If it has not been properly configured, the mlagent executable returns an error code.

- If you get this code, then run the agent in configuration mode to set it up; then try running the agent in normal mode again. See "Configuring and running the MobiLink Agent on the client device" on page 134.

MobiLink Monitor

Introduction to the MobiLink Monitor

The MobiLink Monitor is a MobiLink administration tool that provides you with details about the performance of your synchronizations.

When you start the MobiLink Monitor and connect it to a MobiLink server, the MobiLink Monitor begins to collect statistical information about all synchronizations that occur in that monitoring session. The MobiLink Monitor continues to collect data until you disconnect it or shut down the MobiLink server.

You can view the data in tabular or graphical form in the MobiLink Monitor interface. You can also save the data in binary format for viewing with the MobiLink Monitor later, or in *.csv* format to open in another tool, such as Microsoft Excel; or you can export it to an ODBC data source such as a MobiLink-supported relational database.

MobiLink Monitor output allows you to see a wide variety of information about your synchronizations. For example, you can quickly identify synchronizations that result in errors, or that meet other criteria that you specify. You can identify possible contention in synchronization scripts by checking to see if synchronizations of differing durations have phases that end around the same time (because synchronizations are waiting for a previous phase to finish before they can continue).

The MobiLink Monitor can be used routinely in development and production, because monitoring does not degrade performance, particularly when the MobiLink Monitor is run on a different computer from the MobiLink server.

SQL Anywhere Monitor

The SQL Anywhere Monitor is a browser-based administration tool that provides you with information about the health and availability of SQL Anywhere databases and MobiLink servers. It is useful in assessing overall system health and availability, and for analyzing overall synchronization statistics. The SQL Anywhere Monitor does not provide information about individual synchronizations. For detailed information about individual synchronizations, including timing and other per-synchronization statistics, use the MobiLink Monitor.

For more information about the SQL Anywhere Monitor, see "SQL Anywhere Monitor" [*SQL Anywhere Server - Database Administration*].

Starting the MobiLink Monitor

You can have multiple instances of the MobiLink Monitor running for each MobiLink server.

To start monitoring data

1. Start your consolidated database and MobiLink server, if they are not already running.

2. From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **Administration Tools** » **MobiLink Monitor**.

   Alternatively, you can type **mlmon** at a command prompt. For details, see below.

3. A MobiLink Monitor connection starts like a synchronization connection to the MobiLink server. For example, if you started the MobiLink server with -zu+ then it doesn't matter what user ID you use here. For all MobiLink Monitor sessions, the script version is set to for_ML_Monitor_only.

   The **Connect To MobiLink Server** window should be completed as follows:

   - **Host** The network name or IP address of the computer where the MobiLink server is running. By default, it is the computer where the MobiLink Monitor is running. You can use **localhost** if the MobiLink server is running on the same computer as the MobiLink Monitor.
   - **Protocol** This should be set to the same network protocol that the MobiLink server is using for synchronization requests.
   - **Port** This should be set to the same network port that the MobiLink server is using for synchronization requests.
   - **Encryption** If you chose HTTPS or TLS for the protocol, this box is enabled. You can choose an encryption type from the dropdown list.

     To use HTTPS and TLS, you must have MobiLink client-side data stream encryption installed on the computer running the MobiLink Monitor. For more information about security, see "Encrypting MobiLink client/server communications" [*SQL Anywhere Server - Database Administration*].

     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*].

   - **Trusted Certificate File** If you chose HTTPS or TLS for the protocol, specify the name of the trusted certificate file to be used for secure connections to the MobiLink server. For Windows platforms, the trusted certificate store is used if a trusted certificate file is not supplied. Non-Windows platforms require a trusted certificate file be specified for a secure connection.
   - **Additional Protocol Options** Specify optional network parameters in this field. The allowed values depend on the connection stream type. Multiple parameters should be separated by a semicolon.

     All valid MobiLink client network protocol options are supported, except for those already set in this window, such as host, port and trusted certificate.

     See "MobiLink client network protocol options" [*MobiLink - Client Administration*].

4. Start synchronizing.

   The data appears in the MobiLink Monitor as it is collected.

Starting mlmon on the command line

Command line options allow you to have the MobiLink Monitor open a file or connect to a MobiLink server on startup. Use the following syntax:

mlmon [*connect-options* | *inputfile*.{ **mlm** | **csv** }]

where:

connect-options can be one or more of the following:

- **-c** "*keyword=value*;..."

  The -c option closes the MobiLink Monitor at the end of the connection and saves the session to the specified database. The *keyword=value* pairs provide the connection information for the database that you want the session exported to. For example, `mlmon -c "dsn=mydsn;uid=me;pwd=pwd"`.

- See "Connection parameters" [*SQL Anywhere Server - Database Administration*].

- **-o** *outputfile*.{**mlm**|**csv**}

  The -o option closes the MobiLink Monitor at the end of the connection and saves the session in the specified file.

- **-p** *password*

- **-u** *ml_username* (required to connect to the MobiLink server)

- **-x** {**tcpip**|**tls**|**http**|**https**}[**(***keyword=value*;... **)**]

  The keyword=value pairs can be the host, protocol, and Additional Network Parameters as described above. The -x option is required to connect to the MobiLink server.

You can type **mlmon -?** to view the mlmon syntax.

Starting the MobiLink Monitor on Unix

The following steps can be used if you are using a version of Linux that supports the Linux Desktop icons and if you chose to install them when you installed SQL Anywhere 12.

To start the MobiLink Monitor (Linux Desktop icons)

1. From the **Applications** menu, choose **SQL Anywhere 12** » **Administration Tools** » **MobiLink Monitor**

2. Enter the information to connect to the MobiLink server described in the procedure To start monitoring data.

Note
The following steps assume that you have already sourced the SQL Anywhere utilities. See "Setting environment variables on Unix and Mac OS X" [*SQL Anywhere Server - Database Administration*].

To start the MobiLink Monitor (Unix command line)

1. In a terminal session, enter the following command:

   ```
   mlmon
   ```

2. Enter the information to connect to the MobiLink server as described above.

Stopping the MobiLink Monitor

To stop the MobiLink Monitor

1. In the MobiLink Monitor, choose **Monitor** » **Disconnect From MobiLink Server**. This stops the collection of data.

   You can also stop collecting data by shutting down the MobiLink server or closing the MobiLink Monitor.

   Before closing the MobiLink Monitor, you can save the data for the session. See "Saving MobiLink Monitor data" on page 166.

2. When you are ready to close the MobiLink Monitor, choose **File** » **Close**.

Using the MobiLink Monitor

The MobiLink Monitor has the following panes:

- **Details Table** **Details Table** is the top pane. It is a spreadsheet that shows the total time taken by each synchronization, with a breakdown showing the amount of time taken by each part of the synchronization.

  See "Details Table pane" on page 158.

- **Utilization Graph** **Utilization Graph** is the second pane. It provides a graphical representation of queue lengths for different queues on the MobiLink server. The same scale is used for the **Utilization Graph** pane and **Chart** pane. The scale at the bottom of the **Chart** pane represents time. You can select the data that is displayed in the utilization graph by dragging and selecting the data in the **Overview** pane below, or by choosing **View** » **Go To**.

  See "Utilization Graph pane" on page 159.

- **Chart** **Chart** is the third pane. It provides a graphical representation of synchronizations. The scale at the bottom of this pane represents time. You can select the data that is displayed in the chart by dragging and selecting the data in the **Overview** pane below, or by choosing **View** » **Go To**.

  See "Chart pane" on page 162.

- **Overview** **Overview** is the bottom pane. It shows an overview of all synchronizations in the session. This pane contains a box outline called the **Marquee Tool** that can select the data appearing in the **Chart** and **Utilization Graph** panes.

  See "Overview pane" on page 163.

In addition, there is an **Options** window that you can use to customize the display, and properties windows for viewing more details. See:

- "Options window" on page 164
- "Session properties" on page 164
- "Sample properties" on page 164
- "Synchronization properties" on page 166

Details Table pane

The **Details Table** provides information about the duration of each part of the synchronization. All times are measured by the MobiLink server. Some times may be non-zero even when you do not have the corresponding script defined.

You can choose the columns that appear in the **Details Table** pane by opening **Tools** » **Options** and then opening the **Table** tab. For a description of the statistics that are available, see "MobiLink statistical properties" on page 168.

The following columns appear by default:

- **sync** Identifies each synchronization. This ID is assigned by the MobiLink server not by the MobiLink Monitor, so it does not necessarily start at 1 in any given MobiLink Monitor session and is not received in numerical order. You can see the same IDs in the **Synchronization Properties** window. See "Synchronization properties" on page 166.

- **remote_id** The ID of the remote database.

- **user** The synchronization user.

- **version** The version of the synchronization script.

  See "Script versions" on page 282.

- **download_ack** The type of download ack, which can be none, blocking or non-blocking.

- **start_time** The date and time when the MobiLink server started the synchronization. (This may be later than when the synchronization was requested by the client.)

- **duration** The total duration of the synchronization, in seconds.

- **sync_request** The time in seconds for MobiLink to receive the uploaded data from the client. This is the portion of the synchronization starting at the connection from the remote and ending just before authentication.
- **receive_upload** The time taken between creating the network connection between the remote database and the MobiLink server, up to receiving the first bytes of the upload stream. This time is insignificant unless you have set -sm to a smaller value than -nc, then this time can include the time that a synchronization is paused when the number of synchronizations is larger than the maximum number of active synchronizations that were specified with -sm.
- **get_db_worker** The time required to acquire a free database worker thread.
- **connect** The time required by the database worker thread to make a database connection if a new database connection is needed. For example, after an error or if the script version has changed.
- **authenticate_user** The time in seconds for MobiLink to validate the synchronization request, validate the user name, and validate the password (if your synchronization setup requires authentication). This is the length of the authenticate user transaction (from the start of authentication to just before the begin_synchronization event).
- **begin_sync** The time in seconds to run your begin_synchronization script, if one was run.
- **apply_upload** The time in seconds to apply the upload to the consolidated database. This is the time between the begin_upload script and the end_upload script.
- **prepare_for_download** The time in seconds to run your prepare_for_download script, if one was run.
- **fetch_download** The time in seconds to download the data. This is the time between the begin_download script and the end_download script. If download acknowledgement is enabled, this includes the time to apply the download on the remote database and return acknowledgement.
- **end_sync** The time in seconds to run the end_synchronization script, if one was run.

To sort the table by a specific column, click the column heading. If new data is appearing in the MobiLink Monitor, it gets sorted as it is added.

You can close the **Details Table** pane by clearing **Details Table** option in the **View** menu.

Utilization Graph pane

The **Utilization Graph** is the second pane from the top. It displays server statistics for several types of work queues.

For more information about the data available in this pane, see "Using the Utilization Graph" on page 160.

The **Utilization Graph** uses the same horizontal scrollbar, horizontal time labels, and zoom level as the **Chart**. This means that an instant in time lines up vertically between the **Graph** pane and the **Chart** pane.

There are multiple ways to select the data that is displayed in the graph:

- From the **View** menu, choose **Go To**.
- In the **Overview** pane, move the **Marquee Tool**. The **Marquee Tool** is the small box that appears in the **Overview** pane. See "Marquee tool" on page 163.

You can double-click an area of the **Utilization Graph** to open a **Sample Properties** window that shows the details of the sample interval it represents. The sample interval is about a second long. See "Sample properties" on page 164.

Note
The list of properties is obtained from the MobiLink server or the *.mlm* file that was opened, and it uses the language of the MobiLink server. Some characters may not display properly if the MobiLink Monitor is using a different language.

Using the Utilization Graph

To see the values for the **Utilization Graph**, and to customize the output, choose **Tools** » **Options** and open the **Graph** tab. This tab identifies the **Utilization Graph** queues by color, and allows you to customize the graph.

Properties

- **TCP/IP Work Queue** This queue represents work done by the low-level network layer in the MobiLink server. This layer is responsible for both reading and writing packets from and to the network. The queue is full of read and write requests. It grows when it gets notified of incoming data to read off the network and/or when told by the stream worker to write to the network.

  If this queue gets backed up, it is usually due to a backlog of either reads or writes—sometimes both but usually one or the other. Reads can get backed up if the server is using a lot of RAM and memory pages are being swapped in and out a lot. Consider getting more RAM. Writes can get backed up if the network connection between the clients and server is slow. If this queue is the only queue that is backed up, look at the CPU usage. If CPU usage is high, suspect read/memory problems. If CPU usage is low, you may have slow writes. Consider using a faster network.

- **Stream Work Queue** For version 10 clients only. This queue represents work done by the high-level network layer in the MobiLink server. This layer is responsible for higher-level network protocol work such as HTTP, encryption, and compression. This queue grows when lots of reads come in from the TCP/IP layer and/or when lots of write requests come in from the Command processor layer. If this queue is the only queue that is backed up, consider removing some network protocols, such as HTTP or compression. If this is not possible, consider reducing the number of concurrent synchronizations allowed using the -sm option.

  See "-sm mlsrv12 option" on page 67.

- **Heartbeat Work Queue** This queue represents the layer in MobiLink server that is responsible for sending pulsed events within the server. This layer is responsible, for example, for triggering the one-per-second pulses of samples to the connected MobiLink Monitors.

It is highly unlikely that this queue gets backed up so it is not visible by default.

- **Command Processor Work Queue** This represents work done by the MobiLink server to both interpret internal MobiLink protocol commands and apply these commands to the consolidated database. This queue grows when lots of requests come in. Request types include synchronization requests, Listener requests, mlfiletransfer requests, and so on. The queue also grows when the consolidated database is busy working on synchronizations, yet more synchronization requests keep coming in.

  If this queue is the only queue that is backed up, look at the CPU usage. If CPU usage is high, the volume of requests may be too high. Consider reducing the number of concurrent synchronizations allowed, using the -sm option. If CPU usage is low, look to improving the performance of the consolidated database.

  See "-sm mlsrv12 option" on page 67.

- **Busy Database Worker Threads** This value indicates how hard the MobiLink server is pushing the consolidated database. Each unit in the value represents a database worker thread that is doing something in the database. There is no distinction between inserts, updates, deletes, or selects. When this value is zero, the server is not operating on the consolidated database.

  When this count is high (when it is close to the maximum set with the mlsrv12 -w option), the MobiLink server is pushing the consolidated database as hard as it can. In this case, if your throughput is satisfactory, there is nothing to do. If your throughput is not satisfactory, consider increasing the number of database worker threads via the -w option. Note that a higher -w value leads to greater contention between connections. This is particularly bad when all connections are performing uploads, so you may need to use the mlsrv12 -wu option to set a lower limit for database workers doing uploads. If you cannot seem to find settings for -w and -wu that provide adequate throughput, examine your synchronization scripts for possible contention issues. Finally, you can consult your RDBMS documentation for ways to improve the overall performance of your consolidated database.

- **OE Work Queue** This represents the size of the Outbound Enabler (OE) queue. The queue size is proportional to the network I/O activity between the MobiLink server being monitored and the Relay Server(s).

- **Notifier Work Queue** This represents the size of the Notifier work queue. The queue size increases if the consolidated database is slow to respond with the results of the Notifier's queries.

  The Notifier's queries for Remote Tasks are very simple, but may return many rows in large deployments. The queue size is proportional to the length of time it takes the Notifier to retrieve the Remote Task notifications.

- **Dynamic Cache Work Queue** This queue represents the length of time it takes to perform dynamic cache sizing. This line is hidden by default.

Scale

This column tells you the current scale for each property.

The vertical scale on the **Utilization Graph** always goes from 0 to 100. This represents zero to one hundred percent of the scale. Each value has its own scale. By default, all scales are 5, meaning that

values are expected to be in a range of 0 to 20, scaled (by 5) to the range 0 to 100. If a value becomes greater than 20, the scale is automatically adjusted such that the largest value is at 100.

To determine the maximum value in the display, divide the scale into 100. For example, if the TCP/IP work queue scale is 2.381, then the maximum value is (100 / 2.381) = 42. The actual maximum isn't usually important. What is important is that values towards the top of the graph are approaching the largest currently-known value for the given property—in other words, the peak load for that property as observed in the current monitoring session.

When the graphs are consistently towards the top of the display and you notice that synchronization throughput is down, you may have a performance problem that needs investigation. Similarly, if one or more values creeps upward over time without diminishing, then there is likely a performance problem. note that the graphs may often be towards the top of the display with MobiLink server performing well. This just means that MobiLink server is busy and doing its job well.

Antialiasing

One of your customization choices is antialiasing. Antialiasing makes the graph look better, but can be slower to draw.

Chart pane

The **Chart** pane presents the same information as the **Details Table**, but in graphical format. The bars in the **Chart** represent the length of time taken by each synchronization, with sub-sections of the bars representing the phases of the synchronization.

Viewing data

Click a synchronization to select that synchronization in the **Details Table**.

Double-click a synchronization to open the **Synchronization Properties** window. See "Synchronization properties" on page 166.

Grouping data by remote ID or compactly

You can group the data by user. Choose **View** » **By Remote ID**.

Alternatively, you can view the data in a compact mode that shows all active synchronizations in as few rows as possible. Choose **View** » **Compact View**. In **Compact View**, the row numbers are meaningless.

Zooming in on data

There are several ways to select the data that is visible in the **Chart** pane:

- **Zoom options** There are zoom options in the **View** menu and zoom buttons on the toolbar that allow you to zoom in and out. To have a synchronization fill the available space, use **Zoom To Selection**.

- **Scrollbar** Click the scrollbar at the bottom of the **Chart** pane and slide it.

- **Go To window** To open this window, click **View** » **Go To**.

  Start Date & Time lets you specify the start time for the data that appears in the **Chart** pane. If you change this setting, you must specify at least the year, month, and date of the date-time.

  Chart Range lets you specify the duration of time that is displayed. The chart range can be specified in milliseconds, seconds, minutes, hours, or days. The chart range determines the granularity of the data: a smaller length of time means that more detail is visible.

- **Marquee Tool** In the **Overview** pane, move the **Marquee Tool**. The **Marquee Tool** is the small box that appears in the **Overview** pane. See "Marquee tool" on page 163.

Time axis

At the bottom of the **Chart** pane there is a scale showing time periods. The format of the time is readjusted automatically depending on the span of time that is displayed. You can always see the complete date-time by hovering your cursor over the scale.

Default color scheme

You can view or set the colors in the **Chart** pane by opening the **Options** window (available from the **Tools** menu). The default color scheme for the **Chart** pane uses lime green for uploads, coral red for downloads, and blue for begin and end phases, with a darker shade for earlier parts of a phase.

For information about setting colors, see "Options window" on page 164.

Overview pane

The **Overview** pane shows an overview of the entire MobiLink Monitor session. You can navigate through the session using the **Marquee Tool**, which is the box inside the **Overview** pane.

Active synchronizations, completed synchronizations, and failed synchronizations are represented with colors. To set the colors, open the MobiLink Monitor, choose **Tools** » **Options**, and then click the **Overview** tab.

See "Options window" on page 164.

You can close the **Overview** pane by deselecting it in the **View** menu.

You can also separate the **Overview** pane from the rest of the MobiLink Monitor window. In the **Options** window, open the **Overview** tab and clear the **Keep overview window attached to main window** checkbox.

Marquee tool

The **Marquee Tool** is the small box that appears in the **Overview** pane. You can use the **Marquee Tool** to see different data, or to see data at different granularity. The area represented within the box is displayed in the chart and graph panes. You can use the **Marquee Tool** as follows:

- Click in the **Overview** pane to move the **Marquee Tool** and the start time of the data shown in the chart or utilization graph.

- Drag in the **Overview** pane to redraw the **Marquee Tool** to change the **Marquee Tool**'s location and size and change the start time and the range of data. If you make the marquee box smaller, you shorten the interval of the visible data in the chart, which makes more detail visible.

To change the color of the Marquee Tool

1. Choose **Tools** » **Options**.
2. Click the **Overview** tab.
3. Select a new color in the **Marquee** field.
4. Click **OK**.

Options window

Options allow you to specify many settings, including colors and patterns for the graphical display in the **Chart** pane and the **Overview** pane.

To open the **Options** window, open the MobiLink Monitor and choose **Tools** » **Options**.

Restoring defaults

To restore default settings, delete the file *mlMonitorSettings11*. This file is stored in your user profiles directory.

Session properties

The **Session Properties** window provides statistics about the session. It provides property values for the entire time that the MobiLink Monitor has been running. To open the **Session Properties** window, open the MobiLink Monitor and choose **File** » **Properties**.

The **General** tab provides basic information about the session.

The **Statistics** tab shows the same statistics as **Sample Properties**. See "Sample properties" on page 164.

Sample properties

The **Sample Properties** window provides detailed statistics for time intervals. Each time interval is about one second long. Samples are numbered by the MobiLink Monitor to reflect the order in which they were received.

You can customize the appearance of the graph to hide properties but all properties appear in the **Sample Properties** window. If you have hidden a property, it is identified as **Hidden** in the **Sample Properties** window; otherwise the color is shown.

To open the **Sample Properties** window, click in the **Graph** pane for the time period that you want to examine.

The **Sample** tab provides information for the one-second time interval it represents. The properties that are displayed are for the end of the time interval.

The **Range** tab shows averages for the entire range of samples that were visible when the properties window was opened (the horizontal range that is visible in the **Overview**). The range statistics are not calculated until you click **Calculate** in the **Range** tab.

Sample Properties contains the following information:

- **Sample** Samples are numbered by the MobiLink Monitor to reflect the order in which they were received. On the **Sample** tab, this shows the sample number. On the **Range** tab it shows the range of samples.
- **Start time and end time** On the **Sample** tab, this reflects a sample time period of approximately one second.
- **Statistics - Color column** The color used in the graph for this property.
- **Statistics - Property column** Shows the queue length for the sample or average queue for the range. This column displays the following types of property:
  - **TCP/IP work queue** This lists the number of buffers waiting to be filled by reading from the network plus the number of buffers waiting to be written to the network. The actual number is not very meaningful, but large numbers may indicate network-related bottlenecks.
  - **Stream work queue** For version 10 clients only. This queue represents work done by the high-level network layer in the MobiLink server. This layer is responsible for higher-level network protocol work such as HTTP, encryption, and compression.
  - **Heartbeat work queue** This is the queue length for periodic internal MobiLink server tasks other than synchronizations.
  - **Command processor work queue** This is the queue length for performing database tasks and interpreting or preparing communications with MobiLink clients. The actual number is not very meaningful, but large numbers may indicate database-related bottlenecks.
  - **Busy database worker threads** This value indicates how hard the MobiLink server is pushing the consolidated database. Each unit in the value represents a database worker thread that is doing something in the database. There is no distinction between inserts, updates, deletes, or selects. When this value is zero, the server is not operating on the consolidated database.

  Note: The names of the properties are obtained from the MobiLink server or .mlm file and are in the language of the MobiLink server. Some characters may not display properly if the MobiLink Monitor is using a different language.
- **Statistics - Value column** The property value.

- **Statistics - Limit column** The maximum allowed value for the property. This is useful so that the graph can use a scale of 0-100% for all properties. For properties such as page faults that are essentially unbounded, the limit is ignored.

Synchronization properties

Double-click a synchronization in either the **Details Table** pane or the **Chart** pane to see properties for that synchronization.

You can choose to see statistics for all tables (which is the sum for all tables in the synchronization), or for individual tables. The dropdown list provides a list of the tables that were involved in the synchronization.

For descriptions of the quantities displayed on any page of the **Synchronization Properties** window, click **Help**.

For an explanation of the statistics in **Synchronization Properties** window, see "MobiLink statistical properties" on page 168.

Saving MobiLink Monitor data

You can save the data from a MobiLink Monitor session as a binary file (.mlm), as a text file with comma-separated values (.csv), as tables in a relational database, or as a Microsoft Excel file.

Saving to file

To save the data as a file, choose **File** » **Save As**.

- Save the data as a binary (.mlm) file if you want to view the saved data in the MobiLink Monitor. To reopen, choose **File** » **Open**. The binary file format is the only format that preserves all monitored information.

- Save the data as a comma separated file (*.csv*) if you want to view it in another tool, such as Microsoft Excel. This only saves the information in the synchronization properties windows, except per table information and the session end time. You can also open a .csv file in the MobiLink Monitor.

  In the *.csv* file format, time durations are stored in milliseconds.

You can specify that you want data to be saved automatically to a file. To do this, choose **Tools** » **Options**, and enter an output file name on the **General** tab. The output file is overwritten by new data.

Exporting to a relational database or Excel

You can also export MobiLink Monitor data using an ODBC connection. You can export to any relational database that is supported by MobiLink, and to a Microsoft Excel spreadsheet.

When you export data, all the columns in your MobiLink Monitor session are exported, and a column named export_time that identifies the time the export was performed. Data from the graph is not exported.

The data source must have quoted identifiers enabled, because some of the columns are reserved words. The MobiLink Monitor enables quoted identifiers automatically for SQL Anywhere, Adaptive Server Enterprise, and Microsoft SQL Server databases. If the quoted identifiers option is not enabled, the export fails.

To export the data to a database or Excel

1. After collecting MobiLink Monitor information, disconnect from the MobiLink server.
2. In the MobiLink Monitor, choose **File** » **Export To Database**.
3. Select options for the output.
   - You can name the tables that are created to hold the data, or use the defaults. If the tables do not exist, they are created by the MobiLink Monitor. For Excel output, the table names identify the worksheets that are created.
   - Choose whether you want to overwrite data in existing tables. If you do not choose to overwrite the data, new data is appended to existing data.
4. Click **OK** to open the **Connect** window and connect to the database or Excel spreadsheet using ODBC.

Customizing your statistics

The **Watch Manager** allows you to visibly distinguish synchronizations that meet criteria that you specify. For example, you might want to highlight big synchronizations, long synchronizations, small synchronizations that take a long time, or synchronizations that receive warnings.

To open the **Watch Manager**, open the MobiLink Monitor and then click **Tools** » **Watch Manager**.

The left pane of the **Watch Manager** contains a list of all available watches. The right pane contains a list of active watches. To add or remove a watch from the active list, select a watch in the left pane and click the appropriate button.

There are three predefined watches (**Active**, **Completed**, and **Failed**). You can edit predefined watches to change the way they are displayed, and you can deactivate them by removing them from the right pane.

No synchronizations are displayed in the chart unless they meet the conditions of a watch. If you disable all watches (by removing them from the **Current Watches** list), then no synchronizations are shown in the **Chart** pane or the **Overview** pane.

The order of watches in the right pane is important. Watches that are closer to the top of the list are processed first. Use the **Move Up** and **Move Down** buttons to organize the order of watches in the right pane.

You can use the predefined watches, and create other watches. To edit a watch condition, remove it and then add the new watch condition.

When a new MobiLink Monitor connects to the same MobiLink server, it shows up as a short synchronization in any MobiLink Monitors that are already connected. The MobiLink Monitor

synchronization has the version name for_ML_Monitor_only. You can hide this MobiLink Monitor synchronization with a watch.

To create a new watch

1. In the **Watch Manager**, click **New**.
2. Give the watch a name in the **Name** box.
3. Select a **Property**, comparison **Operator**, and **Value**.

   For a complete list of properties, see "MobiLink statistical properties" on page 168.
4. Click **Add**. (You must click **Add** to save the settings.)
5. If desired, select another **Property**, **Operator**, and **Value**, and click **Add**.
6. Select a **Chart Pattern** for the watch in the **Chart** pane.
7. Select an **Overview Color** for the watch in the **Overview** pane.

MobiLink statistical properties

The following is a list of the properties that are available in the MobiLink Monitor. These statistics can be viewed in the **New Watch** window, the **Details Table** pane, or the **Synchronization Properties** window. In **Synchronization Properties**, the property names do not contain underscores.

For more information about the **New Watch** window, see "Customizing your statistics" on page 167.

For more information about the **Details Table**, see "Details Table pane" on page 158.

For more information about the **Synchronization Properties** window, see "Synchronization properties" on page 166.

Synchronization statistics

MobiLink statistical properties return the following information for synchronizations when not using forced conflict mode.

For information about forced conflict mode, see "Forced conflict statistics" on page 172.

Note
Forced-conflict mode has been deprecated.

Property	Description
active	True if the synchronization is in progress.

Property	Description
apply_upload	The time required for the uploaded data to be applied to the consolidated database.
authenticate_user	Total time to perform user authentication, including executing the authenticate_* events.
begin_sync	Total time for the begin_synchronization event.
completed	True if the synchronization completed successfully.
conflicted_deletes	Always zero.
conflicted_inserts	Always zero.
conflicted_updates	Number of update rows that caused conflict. A row is included only when a resolve conflict script was successfully called for it.
connect	The time required by the database worker thread to make a database connection if a new database connection is needed. For example, after an error or if the script version has changed.
connect_for_download_ack	The time required by the database worker thread to make a database connection if a new database connection is needed.
connection_retries	Number of times the MobiLink server retried the connection to the consolidated database.
download_bytes	Amount of memory used within the MobiLink server to store the download. This provides a good indication of the impact on server memory of a synchronization.
download_deleted_rows	Number of row deletions fetched from the consolidated database by the MobiLink server (using download_delete_cursor scripts).
download_errors	Number of errors that occurred during the download.
download_fetched_rows	Number of rows fetched from the consolidated database by the MobiLink server (using download_cursor scripts).
download_filtered_rows	Number of fetched rows that were not downloaded to the MobiLink client because they matched rows that the client uploaded.
download_warnings	Number of warnings that occurred during the download.

Property	Description
duration	Total time for the synchronization, as measured by the MobiLink server.
end_sync	Total time for the end_synchronization event.
fetch_download	The time required to fetch the rows to be downloaded from the consolidated database to create the download stream.
get_db_worker	The time required to acquire a free database worker thread.
get_db_worker_for_download_ack	The time spent waiting for a free database worker thread after the download acknowledgement has been received.
ignored_deletes	Number of upload delete rows that caused errors while the upload_delete script was invoked, when the handle_error or handle_odbc_error are defined and returned 1000, or when there is no upload_delete script defined for the given table.
ignored_inserts	The total number of upload insert rows that were ignored. They were ignored because 1) there is no upload_insert script in normal mode or no upload_new_row_insert script in forced conflict mode; or 2) errors occurred when the MobiLink server was invoking the corresponding script and the handle_error or handle_odbc_error event returned 1000.
ignored_updates	Number of upload update rows that caused conflict but a resolve conflict script was not successfully called or no upload_update script was defined.
nonblocking_download_ack	The time required for the publication_nonblocking_download_ack connection and nonblocking_download_ack connection events.
prepare_for_download	Total time for the prepare_for_download event.
remote_id	The remote ID that uniquely identifies the remote database.
send_download	The time required to send the download stream to the remote database. The time depends on the size of the download stream and the network bandwidth for the transfer. For an upload-only synchronization, the download stream is simply an upload acknowledgement.
start_time	Date-time (in ISO-8601 extended format) for the start of the synchronization.

Property	Description
sync	A number uniquely identifying the synchronization within the MobiLink Monitor session.
sync_deadlocks	Number of deadlocks in the consolidated database that were detected for the synchronization.
sync_errors	Total number of errors that occurred for the synchronization.
sync_request	The time taken between creating the network connection between the remote database and the MobiLink server, up to receiving the first bytes of the upload stream.
sync_tables	Number of client tables that were involved in the synchronization.
sync_warnings	Number of warnings that occurred for the synchronization.
upload_bytes	Amount of memory used within the MobiLink server to store the upload. This provides a good indication of the impact on server memory of a synchronization.
upload_deadlocks	Number of deadlocks in the consolidated database that were detected during the upload.
upload_deleted_rows	Number of rows that were successfully deleted from the consolidated database.
upload_errors	Number of errors that occurred during the upload.
upload_inserted_rows	Number of rows that were successfully inserted in the consolidated database.
upload_updated_rows	Number of rows that were successfully updated in the consolidated database.
upload_warnings	Number of warnings that occurred during the upload.
user	MobiLink user name.
version	Name of the synchronization version.
wait_for_download_ack	The time spent waiting for the download to be applied to the remote database and for the remote database to send the download acknowledgement.

Forced conflict statistics

When you are in forced conflict mode, MobiLink statistical properties return the following information.

Note
Forced-conflict mode has been deprecated.

Statistical property	Description
conflicted_deletes	Number of upload delete rows that were successfully inserted into the consolidated database using the upload_old_row_insert script.
conflicted_inserts	Number of upload insert rows that were inserted into the consolidated database using the upload_new_row_insert script.
conflicted_updates	Number of upload update rows that were successfully applied using the upload_new_row_insert or upload_old_row_insert scripts.
ignored_deletes	Number of upload delete rows that caused errors while the upload_old_row_insert script was invoked, when the handle_error or handle_odbc_error are defined and returned 1000, or when there is no upload_old_row_insert script defined for the given table.
ignored_inserts	Number of upload insert rows that caused errors while the upload_new_row_insert script was invoked, when the handle_error or handle_odbc_error are defined and returned 1000, or when there is no upload_new_row_insert script defined for the given table.
ignored_updates	Number of upload update rows that caused errors while the upload_new_row_insert or upload_old_row_insert scripts were invoked, when the handle_error or handle_odbc_error are defined and returned 1000, or when there is no upload_new_row_insert and upload_old_row_insert script defined for the given table.
upload_deleted_rows	Always zero.
upload_inserted_rows	Always zero.
upload_updated_rows	Always zero.

SQL Anywhere Monitor for MobiLink

The SQL Anywhere Monitor, also referred to as the Monitor, is a browser-based administration tool that provides you with information about the health and availability of SQL Anywhere databases, MobiLink servers, MobiLink server farms, and Relay Server farms.

This section describes how to use the Monitor to collect metrics about *MobiLink servers* and *MobiLink server farms*. For information about using the Monitor with:

- SQL Anywhere databases, see "SQL Anywhere Monitor" [*SQL Anywhere Server - Database Administration*].
- Relay Server farms, see "SQL Anywhere Monitor for Relay Server" [*Relay Server*].

The Monitor provides the following functionality:

- **Constant data collection** Unlike many of the other administration tools available with SQL Anywhere, the Monitor collects metrics all the time, even when you are not logged in to the browser. The Monitor collects metrics until you shut it down.
- **Email alert notification** As the metrics are collected, the Monitor examines the metrics and can send email alerts when it detects conditions that indicate something is wrong with a resource.
- **Browser-based interface** At any time, you can log in to the Monitor using a browser to review alerts and metrics that have been collected.
- **Monitor multiple databases, MobiLink servers, MobiLink server farms, and Relay Server farms** From one tool, you can simultaneously monitor multiple resources running on the same or different computers.
- **Minimal performance impact** The Monitor can be used routinely in development and production environments because monitoring does not degrade performance.

The Monitor is designed to help any type of user, whether they are a DBA or not, who is responsible for such tasks as:

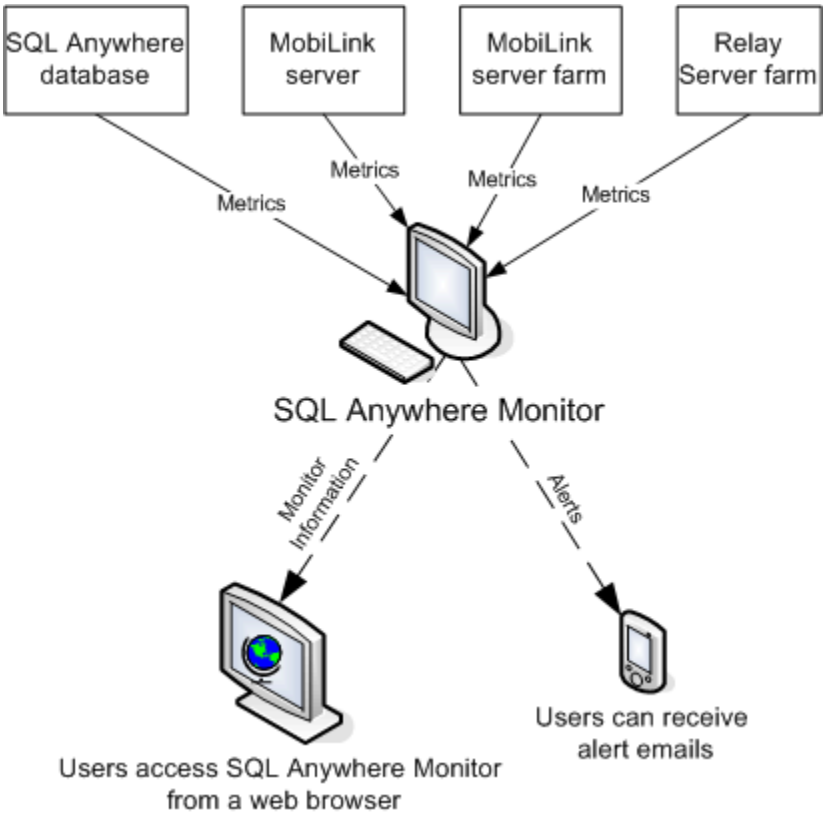
- Ensuring that a MobiLink server is connected to the network.
- Ensuring that there is enough disk space or memory available for a MobiLink server.
- Reviewing the number of synchronizations a MobiLink server performs over a specified time.

See also

For information about other administration and performance tools that are available for MobiLink servers, see: "MobiLink Monitor" on page 154.

Monitor architecture

The Monitor collects metrics and performance data from SQL Anywhere databases, MobiLink servers, MobiLink server farms, and Relay Server farms running on other computers, while a separate computer accesses the Monitor via a browser.



The Monitor is a Flash-based application that is served to a web browser via the SQL Anywhere built-in HTTP server. You interact with the Monitor through its browser interface.

There are two editions of the Monitor:

- **SQL Anywhere Monitor Developer Edition** The Monitor Developer Edition is intended for development and testing use. It is installed by default with SQL Anywhere and it uses the installed SQL Anywhere on the back-end.
- **SQL Anywhere Monitor Production Edition** This Production Edition is intended for deployment and production use. It is installed separately, runs as a service, and includes a fully-contained SQL Anywhere installation. This edition is only available for certain editions of SQL Anywhere. See "Installing the SQL Anywhere Monitor in a production environment" on page 242.

Requirements

For the computer where the Monitor is installed

- To run the SQL Anywhere Monitor on a Windows system that has the Windows Firewall enabled, you must add a port exception for port 4950.
- The SQL Anywhere Monitor reserves 384 MB of virtual address space when you start the Monitor. When you start the Monitor on Linux, you must ensure that you have least this amount available.

- To monitor resources that are secured using ECC encryption or FIPS-certified encryption, you need a separate license. See "Securing the Monitor" on page 244.
- The Monitor can run on the same computer as the resources it is monitoring, but it is recommended, particularly in production environments, that you run the Monitor on a different computer to minimize the impact on the MobiLink server or other applications.
- It is recommended in production environments that you run the Monitor Production Edition. See "Installing the SQL Anywhere Monitor in a production environment" on page 242.
- When running the Monitor Developer Edition, you must have SQL Anywhere 12.0.0 installed. The Monitor Developer Edition uses the installed SQL Anywhere on the back-end.

For the computer accessing the Monitor

- Install the latest version of Adobe Flash Player that is available for your operating system. The Monitor is backwards compatible with version 10 of Adobe Flash Player. To determine the correct version, see http://www.adobe.com/products/flashplayer/systemreqs/.
- Enable JavaScript in your browser.
- Ensure that the computer where you are using a browser to access the Monitor is connected to the network where the Monitor is installed.

  You can access the Monitor from the same computer as it is running on, but it is recommended, particularly in production environments, that you access the Monitor from a different computer.

Limitations

- You can use the Monitor to collect metrics about the following versions of SQL Anywhere databases, MobiLink servers, MobiLink server farms, and Relay Server farms:
  - SQL Anywhere 9.0.2, 10.0.0, 10.0.1, 11.0.0, 11.0.1, and 12.0.0
  - MobiLink 11.0.0 with at least the first EBF applied, 11.0.1, and 12.0.0
  - Relay Server 12.0.0
- You can only run one edition of the Monitor on a computer at a time.
- On Linux, the Monitor Developer Edition can only be run by the user who installed it.
- On Linux, only the user with administrator privilege can install and run the Monitor Production Edition.
- The Monitor does not provide information about individual synchronizations. For detailed information about individual synchronizations, including timing and other per-synchronization statistics, use the MobiLink Monitor. See "Introduction to the MobiLink Monitor" on page 154.

See also

- "Monitor quick start" on page 176

Monitor quick start

The following steps are required to set up MobiLink server and MobiLink server farm monitoring:

1. Start the MobiLink server that you want to monitor (if it is not already running).
2. Install the Monitor on a computer that is always connected to your network.

   The Monitor can run on the same computer as the resources it is monitoring, but it is recommended, particularly in production environments, that you run the Monitor on a different computer to minimize the impact on the MobiLink server or other applications.
3. Start the Monitor and open it in your browser. See "Start the Monitor" on page 185.
4. Log in to the Monitor as an administrator. The default user is an administrator user with the name **admin** and the password **admin**.

   Note
   You must be logged in to the Monitor as an administrator to perform the following tasks. See "Monitor users" on page 230.
5. As an administrator, in the left navigation menu, choose **Tools** » **Administration** and add a MobiLink server or a MobiLink server farma Relay Server farm as a resource to be monitored. See "Add resources" on page 197.
6. As an administrator, add new users and change the password for the admin user. See "Create Monitor users" on page 231.
7. As an administrator, configure alert thresholds for the resources. See "Alerts" on page 234.
8. As an administrator, configure the backup and maintenance schedule. See "Back up the Monitor" on page 240.
9. Click **Close** to exit **Administration**.
10. Click **Overview**. In the **Resource List** widget, you will see the resources that are being monitored. See "Overview dashboard" on page 190.

Tutorial: Using the Monitor

Use this tutorial to set up monitoring of a MobiLink server and a MobiLink server farm. This tutorial uses the Monitor Developer Edition.

Lesson 1: Log in to the Monitor as the default administrator

To start and log in to the Monitor

1. Start the Monitor. The following steps assume that the Monitor is not currently running in the background.

   To start the Monitor Developer Edition (Windows): Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **SQL Anywhere Monitor**.

   To start the Monitor Developer Edition (Linux): Run the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory. Run the following:

   ```
   samonitor.sh launch
   ```

   The Monitor starts collecting metrics and a browser opens the default URL where you can log in to the Monitor: *http://localhost:4950*.

   Note
   If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running.

2. Log in to the Monitor as the default *administrator* user.

   In the **User Name** field, type **admin**, and in the **Password** field, type **admin**.

   Note
   You must be logged in to the Monitor as an administrator to perform the following lessons in the tutorial. Read-only and operator users do not have permission to perform all the tasks.

   To check your Monitor user type

   1. Log in to the Monitor.
   2. Choose **Tools** » **User Settings** and review the **User Type** setting.

   See "Monitor users" on page 230.

By default, the Monitor opens the **Overview** dashboard that contains two widgets:

1. The **Resource List** widget lists the resources that are being monitored. When you first open the Monitor, it is only monitoring itself via the default resource, named **SQL Anywhere Monitor**. You cannot modify this resource, nor can you stop monitoring it.
2. The **Alerts List** widget lists any alerts from the monitored resources.

See also

- "Lesson 2: Set up the Monitor to monitor a MobiLink server" on page 178
- "Start the Monitor" on page 185

Lesson 2: Set up the Monitor to monitor a MobiLink server

In this lesson, you start the **MobiLink Synchronization Server Sample**, and then add the server as a resource to be monitored.

To add a resource to the Monitor

1. Start the MobiLink Synchronization Server Sample.

   From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **MobiLink** » **Synchronization Server Sample**.

2. Log in to the Monitor as an administrator. See "Lesson 1: Log in to the Monitor as the default administrator" on page 177.

3. In the left navigation menu choose **Tools** » **Administration**.

4. Select **Resources**, and then click **Add**.

5. Select **MobiLink Server**, and then click **Next**.

6. In the **Name** field, type **MobiLinkServerSample**, and then click **Next**.

7. In the **Host** field, type **localhost**, and then click **Next**.

8. When you are prompted for the required authorization, in the **User ID** field, type a user name such as **monitor_user**, and in the **Password** field, type a password, such as **sql**.

   These credentials are used to create a user on the MobiLink server. The Monitor stores this user ID and password and uses it to connect to the MobiLink server and monitor it.

9. Click **Create**.

10. The new resource, **MobiLinkServerSample**, is created and monitoring starts.

11. Click **OK**.

12. Click **Close**.

13. Choose **Overview** » **Resource List**. Click **MobiLinkServerSample** to create and open a dashboard for the resource.

See also

- To collect metrics from a SQL Anywhere database, see "Lesson 2: Set up the Monitor to monitor a database" [*SQL Anywhere Server - Database Administration*].
- To collect metrics from a Relay Server farm, see "Lesson 2: Set up the Monitor to monitor a Relay Server farm" [*Relay Server*].

Lesson 3: Set up the Monitor to monitor a MobiLink server farm

In this lesson, you add a MobiLink server farm resource to monitor two MobiLink servers. A MobiLink server farm resource consists of one or more MobiLink server resources.

To add a MobiLink server farm resource

1. Add two MobiLink servers as resources to be monitored. For the first resource, use the **MobiLinkServerSample** resource that you added in the previous lesson. See "Lesson 2: Set up the Monitor to monitor a MobiLink server" on page 178.

   Add a second MobiLink server resource:

   a. At a command prompt, run the following to start a MobiLink server that listens on port 8039:

      ```
      "C:\Program Files\SQL Anywhere 12\Bin32\mlsrv12.exe" -vcrs -zu+ -c
      "dsn=SQL Anywhere 12 CustDB;uid=ml_server;pwd=sql" -ot ml_tcpip.txt -zs
      ml_tcpip -x tcpip{port=8039}
      ```

   b. Log in to the Monitor as an administrator. See "Lesson 1: Log in to the Monitor as the default administrator" on page 177.

   c. Choose **Tools** » **Administration**.

   d. Click **Resources**, and then click **Add**.

   e. Select **MobiLink Server**, and then click **Next**.

   f. In the **Name** field, type **ml_tcpip**, and then click **Next**.

   g. In the **Host** field, type **localhost**.

      In the **Port** field, type **8039**, and then click **Next**.

   h. When you are prompted for the required authorization, in the **User ID** field, type a user name such as **monitor_user**, and in the **Password** field, type a password, such as **sql**.

These credentials are used to create a user on the MobiLink server. The Monitor stores this user ID and password and uses it to connect to the MobiLink server and monitor it.

i. Click **Create**.

The **ml_tcpip** resource is added to the **Resource List** in the **Overview** dashboard.

j. Click **OK**.

k. Click **Close**.

2. Add the MobiLink server farm resource.

a. Open the **Administration** window.

Choose **Tools** » **Administration**.

b. Click **Resources**, and then click **Add**.

c. Select **MobiLink Server Farm**, and then click **Next**.

d. In the **Name** field, type **MobiLink_Test_Farm**, and then click **Next**.

e. Select **MobiLinkServerSample** and **ml_tcpip**, and then click **Create**.

f. Click **OK**.

g. Click **Close**.

3. Choose **Dashboards** » **Overview**.

The **MobiLink_Test_Farm** resource appears in the **Resource List**.

Note that the MobiLink server resources remain in the **Resource List**.

4. Click the arrow to the left of the **MobiLink_Test_Farm** to see the list of MobiLink server resources that are included in the farm.

5. Click on **MobiLink_Test_Farm** to open the **MobiLink_Test_Farm** dashboard and view the collected metrics.

The **Alerts List**, **Resource Widget**, and **Server Info** widgets should appear.

Lesson 4: Test an alert

In this lesson, you intentionally trigger an alert so you can practice handling alerts.

To view and resolve an alert

1. Trigger an alert by shutting down the resource.

For example:

a. On Windows, double-click the MobiLink server icon in the system tray for the **MobiLinkServerSample** MobiLink server.

b. Click **Shut Down** in the MobiLink server messages window.

c. Click **Yes**.

2. Log in to the Monitor as an administrator. See "Lesson 1: Log in to the Monitor as the default administrator" on page 177.

3. Choose **Dashboards** » **Overview**.

   In the **Resource List**, the **Status** for the **MobiLinkServerSample** resource changes to a red circle with a white x through it. This icon indicates that the resource is unavailable.

   In the **Alerts** widget, an **Availability Alert** appears and its status is **Active**.

   It can take a few seconds for these changes in state and status to occur. By default, the Monitor collects information from the resource every 30 seconds. See "Collection intervals" on page 207.

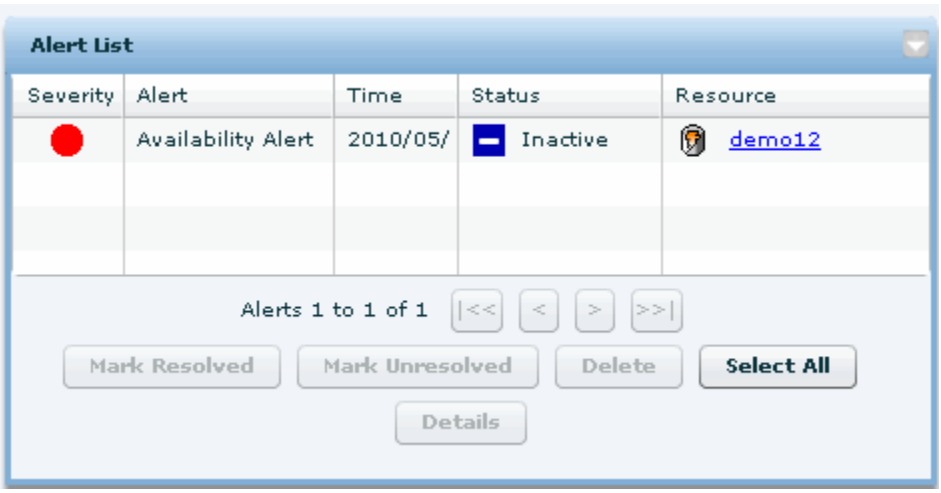
4. In the **Alerts** dashboard, click the **Availability Alert**, and then click **Details** to read the description.

5. Click **OK** to close the alert.

6. Restart the Synchronization Server Sample.

   From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **MobiLink** » **Synchronization Server Sample**.

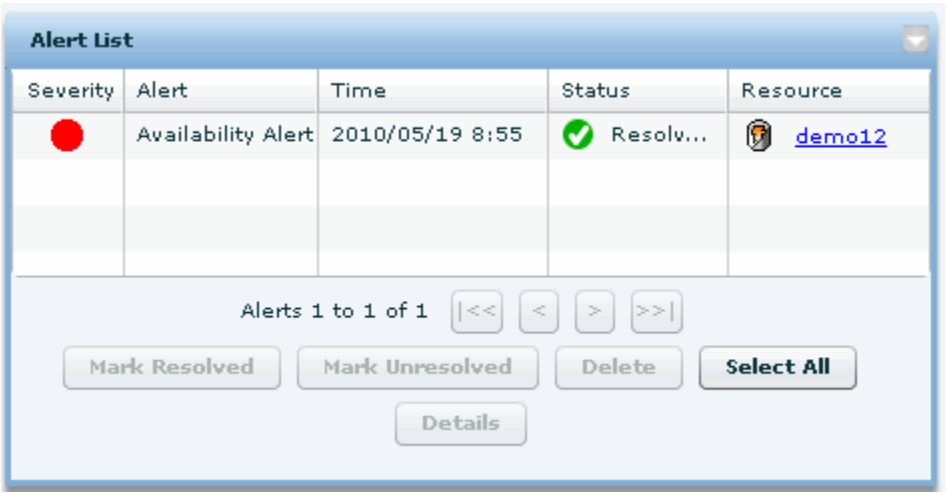
   In the Monitor, in the **Resource List**, the **Status** for the **MobiLinkServerSample** resource changes to a yellow triangle. This icon indicates that the resource is being monitored and it has an alert. In the **Alerts List**, the **State** of the **Availability Alert** changes to **Inactive**. An inactive alert indicates that the issue that triggered the alert is no longer present, but the alert has not been resolved or deleted.



7. Resolve the alert by selecting the alert and clicking **Mark Resolved**.

> **Note**
> Only administrators and operators can resolve and delete alerts. See "Monitor users" on page 230.

Now in the **Resource List**, the **Status** for the **MobiLinkServerSample** resource is blank. No icon in the **Status** column indicates that the resource is being monitored and it has no alerts. In the **Alerts List**, the **State** of the **Availability Alert** changes to **Resolved by *admin*** where *admin* is your Monitor user name.



8. Delete the alert by selecting the alert and clicking **Delete**.

   When prompted, click **Yes** to confirm the deletion.

Lesson 5: Set up the Monitor to send emails when alerts occur

When an alert occurs, it is always listed in the **Alerts List** widget on the dashboard for the particular resource. See "Lesson 4: Test an alert" on page 180.

In this lesson, you set up the Monitor to send you an email whenever an alert occurs.

To set up email notification

1. Log in to the Monitor as an administrator. See "Lesson 1: Log in to the Monitor as the default administrator" on page 177.

2. Add a user that can receive emails.
   a. Choose **Tools** » **Administration**.
   b. Choose **Users**, and then click Add.
   c. In the User Name field, type **JoeSmith**.
   d. In the **Password** and the **Confirm Password** fields, type **password**.
   e. In the **Email** field, enter a valid email address.
   f. For the **User Type**, select **Operator**.

An operator can receive alerts via email and can resolve and delete alerts. This user can access most of the Monitor widgets but it cannot not access the **Administration** window.

g. Click **Next**.

h. When prompted to choose the resources you are interested in, click **Check All**.

i. Click **Save**.

The new user is created and you are returned to the **Administration** window.

3. Configure email alert notification.

> **Note**
> You must be logged in to the Monitor as an administrator to perform the following tasks. Only administrators can configure the Monitor to send emails. See "Monitor users" on page 230.

a. In **Administration** window, select **Configuration** and click **Edit**.

b. On the **Alert Notifications** tab, select **Send Alert Notifications By Email**.

c. Configure other settings as required. See "Enable the Monitor to send alert emails" on page 239.

d. Test that you have properly configured email notification.

Click **Send Test Email**.

e. When prompted, enter an email address to send the test email to and click **OK**.

A test email is sent to the email address specified.

f. Click **Save**.

g. Click **Close**.

When an alert occurs, an email is sent to the specified user with information about the alert.

Lesson 6: Add a new dashboard and widgets

Dashboards are specific to each user. Any user can add, edit or delete their own dashboards. In addition, any user can add, edit, or delete the widgets that exist in their dashboards. By default, when a dashboard is created, it is populated with default widgets. In this lesson, you add a dashboard to the Monitor, and then add widgets.

To add a new dashboard

1. Log in to the Monitor.
2. Choose **Dashboards** » **Add new**.
3. Select **A Dashboard To Monitor The Following Resource**, and choose the resource.
4. In the **Dashboard Name** field, type **user_joe**.
5. In the **Number Of Columns** field, type **2**.

6. Click **OK**.

   A new dashboard appears, with the 4 following widgets: **Alert List**, **Key Performance Metrics**, **Server Info**, and **Connections**.

The following procedure describes how to create a new metrics widget that displays graphs, instead of the default spark lines.

To add a metrics display widget with graphs

1. In the upper right corner of the **user_joe** dashboard, click **Customize**, and then click **Add Widget**.
2. Select **Metrics**, and then click **Next**.
3. In the **What Do You Want To Name This Widget?** field, type **Metrics display**.
4. In the **Which Resource Are You Interested In?** field, choose **MobiLinkServerSample**.
5. In the **What Kind Of Display Do You Want To See?** field, choose **Graph**.
6. For **Which metrics do you want to see?**, select **CPU Usage**, **Memory Metrics** » **Cache Size**, and **Queries Processed**.
7. Click **Create**.

   A widget called **Metrics display** appears in the dashboard.

   To maximize the size of a widget, in the widget pane, click the dropdown menu arrow, and choose **Maximize**.

   To view details on the graph, position your cursor above specific points on the graph.

See also

- "Dashboards" on page 192
- "Widgets" on page 193

Lesson 7: Cleanup

In this lesson, you remove the MobiLinkServerSample resource, which deletes the collected metrics and stops data collection. In a production environment when you want to continue monitoring your MobiLink server, you leave both the MobiLink server and the Monitor running.

> **Note**
> You must be logged in to the Monitor as an administrator to perform the following tasks. Only administrators can remove resources. You cannot delete the **SQL Anywhere Monitor** resource. See "Monitor users" on page 230.

To stop monitoring a resource

1. Log in to the Monitor as an administrator.
2. Remove the **MobiLinkServerSample** resource.
   a. Choose **Tools** » **Administration**.
   b. Select **Resources**.
   c. Select the **MobiLinkServerSample** resource, and click **Stop**.
   d. Click **Remove**.
   e. Click **Yes** to confirm that you want to remove the resource.
   f. Click **Close**.
3. Log out of the Monitor.

   Click **Logout**.
4. Close the browser window where you are viewing the Monitor.
5. Exit the Monitor and stop monitoring.
   - On Windows, in the system tray, right-click the SQL Anywhere Monitor icon and choose **Exit SQL Anywhere Monitor**.
   - On Linux, run the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory:

     ```
     samonitor.sh stop
     ```

   The Monitor stops collecting metrics for all resources.
6. Shut down the MobiLink server.
   a. Double-click the MobiLink Server icon in the system tray for the MobiLink Synchronization Server Sample.
   b. Click **Shut Down** in the MobiLink server messages window.
   c. Click **Yes**.

Start the Monitor

When you start the Monitor, it connects to the resources and collects metrics for *all* resources in the Monitor.

The Monitor is designed to run silently in the background. You interact with the Monitor through its browser interface. It is recommended that you leave the Monitor running continuously in the background to collect the metrics.

Note
You can only run one edition of the Monitor on a computer at a time.

To start the Monitor Developer Edition (Windows)

1. On the computer where the Monitor is installed, choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **SQL Anywhere Monitor**.

   The Monitor connects to the resources and begins collecting metrics for all resources in the Monitor, and:

   - A browser opens the default URL for logging in to the Monitor: *http://localhost:4950*.

     Note
     If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running.

     See "Log in remotely to the Monitor" on page 189.

   - The SQL Anywhere Monitor icon appears in the system tray.

2. Log in.

   In the browser, enter your user name and password for the Monitor. The user name and password for the Monitor are case sensitive. There are three different types of users: administrators, operators, and read-only users. Each type of user has different permissions. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 230.

   You can select **Remember Me On This Computer** to have your session persist for two weeks or until you log out. If **Remember Me On This Computer** is cleared, your session expires when you close the browser or log out.

To start the Monitor Developer Edition (Linux)

1. On the computer where the Monitor is installed, start the Monitor.

   Run the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory:

   ```
   samonitor.sh launch
   ```

   The Monitor connects to the resources and begins collecting metrics for all resources in the Monitor, and a browser opens the default URL for logging in to the Monitor: *http://localhost:4950*.

   Note
   If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 189.

2. Log in.

   In the browser, enter your user name and password for the Monitor. The user name and password for the Monitor are case sensitive. There are three different types of users: administrators, operators, and read-only users. Each type of user has different permissions. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 230.

You can select **Remember Me On This Computer** to have your session persist for two weeks or until you log out. If **Remember Me On This Computer** is cleared, your session expires when you close the browser or log out.

To start the Monitor Production Edition (Windows)

1. On Windows, by default, the Monitor Production Edition runs automatically as a service when installed. Because it runs as a service, this Monitor also starts automatically when the computer starts. If you stop the Monitor, you can restart the Monitor service by doing one of the following steps:
   - On the computer where the Monitor is installed, choose **Start** » **Programs** » **SQL Anywhere 12 Monitor** » **SQL Anywhere Monitor**.
   - Run the *samonitor* batch file from the *bin32* or *bin64* directory in the Monitor installation directory. This batch file starts the Monitor service:

     ```
     samonitor.bat start
     ```

   The Monitor starts collecting metrics.

2. Browse to the URL for logging in to the Monitor. The default URL is *http://localhost:4950*.

   Note
   If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 189.

3. Log in.

   When prompted, enter your user name and password for the Monitor. The user name and password for the Monitor are case sensitive. There are three different types of users: administrators, operators, and read-only users. Each type of user has different permissions. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 230.

   You can select **Remember Me On This Computer** to have your session persist for two weeks or until you log out. If **Remember Me On This Computer** is cleared, your session expires when you close the browser or log out.

To start the Monitor Production Edition (Linux)

1. On Linux, by default, the Monitor Production Edition runs automatically as a service when installed. Because it runs as a service, this Monitor also starts automatically when the computer starts. If you stop the Monitor, you can restart the Monitor service by running the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory. This script starts the Monitor service:

   ```
   samonitor.sh launch
   ```

   The Monitor starts collecting metrics and the browser opens.

2. Browse to the URL for logging in to the Monitor. The default URL is *http://localhost:4950*.

Note
If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 189.

The Monitor connects to the resources and begins collecting metrics for all resources in the Monitor, and a browser opens the default URL for logging in to the Monitor: *http://localhost:4950*.

Note
If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 189.

3. Log in.

   When prompted, enter your user name and password for the Monitor. The user name and password for the Monitor are case sensitive. There are three different types of users: administrators, operators, and read-only users. Each type of user has different permissions. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 230.

   You can select **Remember Me On This Computer** to have your session persist for two weeks or until you log out. If **Remember Me On This Computer** is cleared, your session expires when you close the browser or log out.

See also

- "Stop the Monitor" on page 188
- "To stop the Monitor Production Edition" on page 189
- "Log out from the Monitor" on page 190
- "Overview dashboard" on page 190

Stop the Monitor

Stopping the Monitor stops the collection of metrics for all resources.

Caution
In most cases, it is recommended that you leave the Monitor running, but close the browser. Closing the browser does not stop the collection of metrics.

To stop monitoring a specific MobiLink server or MobiLink server farm see "Stop monitoring resources" on page 204.

To stop the Monitor Developer Edition (Windows)

- In the system tray, right-click the SQL Anywhere Monitor icon and choose **Exit SQL Anywhere Monitor**.

The Monitor stops collecting metrics for all resources.

To restart the Monitor, see "Start the Monitor" on page 185.

To stop the Monitor Developer Edition (Linux)

- Run the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory:

  ```
  samonitor.sh stop
  ```

  The Monitor stops collecting metrics for all resources.

  To restart the Monitor, see "Start the Monitor" on page 185.

To stop the Monitor Production Edition

By default, the Monitor Production Edition runs automatically as a service when installed on Windows and Linux. To stop the Monitor and the collection of all resource metrics, you must stop the service.

To stop the Monitor service (Windows)

- To stop the Monitor service, run *samonitor.bat* from the *bin32* directory in the Monitor installation directory:

  ```
  samonitor.bat stop
  ```

  This batch file stops the Monitor service. The Monitor stops collecting metrics for all resources.

  To restart the Monitor service, see "Start the Monitor" on page 185.

To stop the Monitor service (Linux)

- To stop the Monitor service, run the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory:

  ```
  samonitor.sh stop
  ```

  This script stops the Monitor service. The Monitor stops collecting metrics for all resources.

  To restart the Monitor service, see "Start the Monitor" on page 185.

See also

- "Start the Monitor" on page 185
- "Log in remotely to the Monitor" on page 189
- "Log out from the Monitor" on page 190
- "Overview dashboard" on page 190

Log in remotely to the Monitor

The computer that you are using to log in to the Monitor must be connected to the network where the Monitor is running.

To log in to the Monitor

1. On the computer where the Monitor is installed, start the Monitor. See "Start the Monitor" on page 185.

2. On the computer that is accessing the Monitor, browse to the default URL for logging in to the Monitor: **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. For example, *http://samplehost:4950*.

3. When prompted, enter your user name and password for the Monitor. The user name and password for the Monitor are case sensitive. See "Monitor users" on page 230.

See also

- "Start the Monitor" on page 185
- "Stop the Monitor" on page 188
- "Log out from the Monitor" on page 190
- "Overview dashboard" on page 190

Log out from the Monitor

Logging out from the Monitor has *no* effect on the collection of metrics. If you want to stop collecting metrics, then:

- Stop monitoring the resource. See "Stop monitoring resources" on page 204.

- Exit the Monitor. See "Stop the Monitor" on page 188.

To log out from the Monitor

- Click **Logout** in the upper right corner.

  If you select **Remember Me On This Computer** when you log in to the Monitor, then closing the browser does not log you out of the Monitor.

See also

- "Start the Monitor" on page 185
- "Stop the Monitor" on page 188
- "Log in remotely to the Monitor" on page 189
- "Overview dashboard" on page 190

Overview dashboard

The **Overview** dashboard provides an overview of the health and availability of the resources (for example, MobiLink servers and MobiLink Server Farms) being monitored.

By default, the **Overview** dashboard contains the **Resource List** widget and the **Alert List** widget.

1: Resource List widget

The **Resource List** widget contains a table that lists the resources being monitored, as well as an indication about the overall health of each resource. The table always contains the default resource, named **SQL Anywhere Monitor**, which reports on the health of the Monitor itself. You cannot modify the **SQL Anywhere Monitor** resource, nor can you stop monitoring it.

To view detailed information about a resource

- Click the resource name in the **Resource List** widget.

  The dashboard for the selected resource opens. See "Dashboards" on page 192.

In the **Resource List** widget, the **Status** column provides information about the connections between the Monitor and its resources. The **Status** column indicates whether the resource requires someone to perform an action on it.

A resource has one of the following statuses:

Icon	Status	Description
No icon present	**Healthy**	There are no unresolved alerts for the resources.
	**Alerts Present**	There are one or more unresolved alerts for the resource.
	**Unavaila-ble**	The resource is unavailable. For example, the resource is down.

Icon	Status	Description
	**Monitoring Stopped**	The resource is not being monitored because of a blackout or because a user manually stopped monitoring the resource.

2: Alert List widget

The **Alert List** widget contains the alerts for the monitored MobiLink servers and MobiLink server farm.

To view detailed information about an alert

- Select the alert in the **Alert List**, and then click **Details**.

  A window opens showing the details of the alert.

An alert has one of the following statuses:

Icon	Status	Description
	Active	Active alerts are alerts where the alert condition still applies. No one has resolved the alert.
	Inactive	An inactive alert indicates that the issue that triggered the alert is no longer present, but the alert has not been resolved or deleted.
	**Resolved**	An administrator or operator has marked the alert resolved.

An alert has one of the following levels of severity:

Icon	Severity	Description
	High severity	High severity alerts indicate problems that require a user's immediate attention. For example, when a resource exceeds the low disk space threshold, a high severity alert is issued.
	Medium severity	Medium severity alerts indicate problems that require a user's attention as the problems could escalate. For example, when a resource exceeds the CPU usage threshold, a medium severity alert is issued.
	Low severity	Low severity alerts indicate problems. For example, when a resource has a failed connection, a low severity alert is issued.

See “Alerts” on page 234.

Dashboards

Dashboards are specific to each user. Any user can add, edit, or delete their dashboards.

To add a new dashboard

1. Log in to the Monitor.
2. In the **Dashboards** pane, click **Add New**.
3. Follow the instructions in the window to add a dashboard.
4. Click **OK**.

   The Monitor creates and opens the new dashboard. The name of the dashboard also appears under the **Dashboards** list on the left navigation menu.

To edit or delete a dashboard

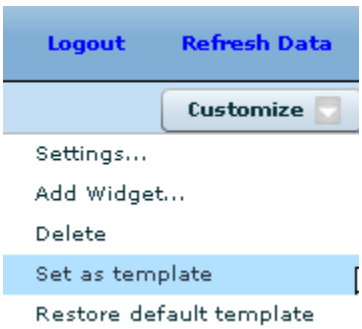
1. Open the dashboard.
2. In the upper right corner of the dashboard, click **Customize**, and then choose either:
   - **Settings** Edits the dashboard's settings.
   - **Delete** Deletes the dashboard.

Dashboard templates

By default, when a dashboard is created, it is populated with default widgets. You can change the default widgets that appear with a new dashboard by configuring the Monitor to use a specified dashboard's widgets and layout. See "Widgets" on page 193.

To create a dashboard template

1. Open the dashboard that you want to use as the template.
2. In the upper right corner of the dashboard, click **Customize**, and then click **Set As Template**.



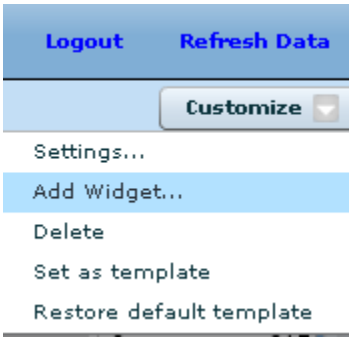
   Now when you add a new dashboard, it contains the same widgets and layout as the template dashboard.

Widgets

Any user can add, edit, or delete the widgets that exist in their dashboards. By default, when a dashboard is created, it is populated with default widgets.

To add widgets

1. Open the dashboard.
2. In the upper right corner of the dashboard, click **Customize**, and then click **Add Widget**.



3. Specify a name for the widget in the **What Do You Want To Name This Widget** field.
4. Specify the resource in the **Which Resource Are You Interested In** field.
5. Choose one of the following display types for the **What Kind Of Display Do You Want** field.
   - **Table display** The table display provides a general outline of the relative values. This display type provides sparklines—simple graphs that are good for showing trends and variations. The table display is the default display type.
   - **Graph display** The graph display is more detailed and is useful for determining exact values at specific times. For example, you notice an unusual peak in a sparkline. To find out more information, such as when a peak occurred, add a widget that uses the graph display to display one or two metrics.
6. Follow the instructions in the **Add Widget** window to add a widget.

   For information about a specific metric, see "List of metrics" on page 209.
7. Click **Create**.

To edit or delete a widget

- In the widget pane, click the dropdown arrow in the upper right corner of the widget, and then choose either:
  - **Settings** Edits the widget's settings.
  - **Delete** Deletes the widget.

See also

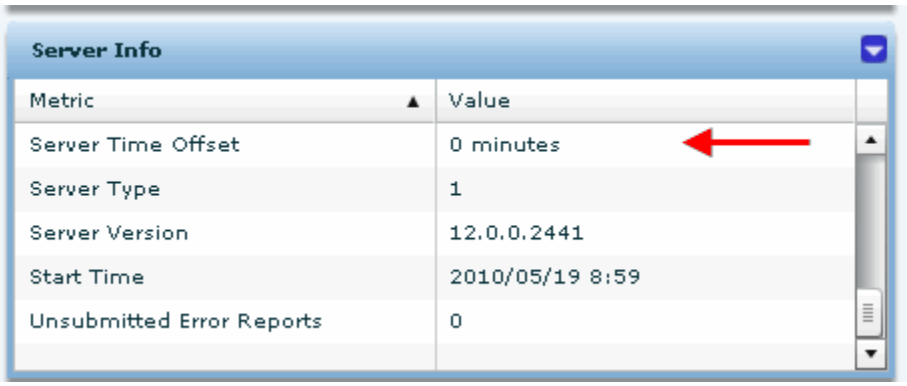
- "Dashboards" on page 192
- "Overview dashboard" on page 190

Understanding how time is displayed

All times displayed in the Monitor are based on the 24-hour clock and are local to the time on the computer that the Monitor is running on.

To find the time difference between the Monitor and your browser

1. Log in to the Monitor.
2. Open the dashboard for the resource.
3. In the **Server Info** widget, find the **Server Time Offset** metric.



The **Server Time Offset** records the time difference between the time on the computer that the Monitor is running and the time on the computer that you are using to view the Monitor data.

See also

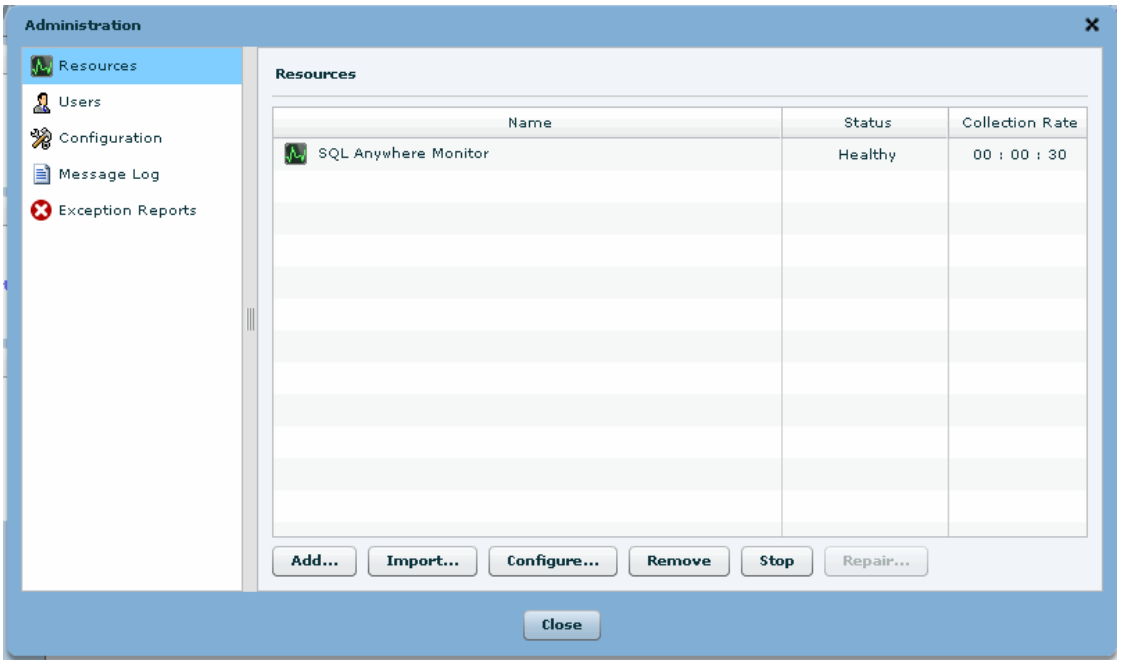
- "Metrics" on page 207

Administration window

Note
Only administrators can access the **Administration** window. For information about administrators, see "Monitor users" on page 230.

As an administrator, you can select the resources (for example, MobiLink servers and MobiLink Server Farms) that you want to monitor, and:

- Add, edit, and delete resources. See “Resources” on page 197.
- Add and edit users. See “Monitor users” on page 230.
- Configure the Monitor. See “Back up the Monitor” on page 240.
- View the Message Log. See “Message Log” on page 196.
- View the Exception Reports. See “Exception Reports” on page 197.



See also

- “Metrics” on page 207

Message Log

The **Message Log** contains informational messages from and about the SQL Anywhere Monitor regarding its operation. Messages are displayed in a table with the most recent message at the top.

To view the message log

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Click **Message Log**.

Exception Reports

When the SQL Anywhere Monitor experiences a fatal error or a crash occurs, an exceptions report is created about what was happening at the time of the problem.

To view the exception reports

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Click **Exception Reports**.

Resources

A **resource** is a SQL Anywhere database, a MobiLink server, a MobiLink Server Farm, or a Relay Server farm. As an administrator, you add resources to the Monitor, and then you start monitoring them.

Monitoring of a resource starts:

- Automatically when the Monitor starts. When you start the Monitor, by default, it connects to the resources and collects metrics for *all* resources in the Monitor.
- Automatically when an administrator adds a resource. After a resource is added, the Monitor attempts to connect to the resource and starts monitoring it. See "Add resources" on page 197.
- Automatically at the end of a blackout period. The Monitor automatically attempts to connect to the resource and resume monitoring.
- When an administrator opens the **Administration** window, clicks **Resources**, selects a resource from the list, and clicks **Start**.

SQL Anywhere Monitor resource

The default resource, named **SQL Anywhere Monitor**, reports on the health of the Monitor itself. This default resource is useful for monitoring the computer that the Monitor is running on and sending alerts when the Monitor is experiencing problems. You cannot modify this resource, nor can you stop monitoring it.

See also

- "Resource List widget" on page 191

Add resources

To monitor a MobiLink server or a MobiLink server farm, an administrator must first add the resource to the Monitor. By default, after the resource is added, monitoring starts.

As an administrator, you can add a resource one at a time or you can add multiple resources via an Import file. See "Add multiple resources" on page 199.

To add a MobiLink server resource to monitor

1. Log in to the Monitor as an administrator.
2. Click **Administration**.
3. Click **Resources**.
4. Click **Add**.
5. Follow the instructions in the **Add Resource** window to add a resource to monitor a MobiLink server or a MobiLink server farm. In the **Host** field, specify 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. This term is required. See "Host connection parameter" [*SQL Anywhere Server - Database Administration*].

   When you add the resource, you must supply a MobiLink user ID and password to connect to the MobiLink server. These credentials are used to connect to the MobiLink server. The MobiLink user ID and password are kept by the Monitor.
6. Click **Create**.

   The resource is added and monitoring of the resource starts.
7. Click **OK**.
8. Click **Close**.

   The resource appears in the **Overview** dashboard **Resource List**.
9. Optional: Add a dashboard for the resource. By default, when a resource is added, a dashboard is not.
   a. Open the **Overview** dashboard.

      Click **Dashboards** » **Overview**.
   b. In the **Resource List**, click the new resource.

      The Monitor creates and opens a new dashboard for the resource. The name of the new dashboard appears in the **Dashboards** section of the sidebar.

      See "Dashboards" on page 192.

To add a MobiLink server farm resource to the Monitor

1. Log in to the Monitor as an administrator.
2. Click **Administration**.
3. Click **Resources**.

4. Click **Add**.
5. Follow the instructions in the **Add Resource** window to add a resource to monitor a MobiLink server farm.

   The Monitor collects metrics about the MobiLink server farm as well as each server that is part of the farm. When you add a MobiLink server farm as a resource, you must specify the servers that are part of the farm and if required add the servers as resources.
6. Click **Create**.

   The resource is added and monitoring of the resource starts.
7. Click **OK**.
8. Click **Close**.

   The resource appears in the **Overview** dashboard **Resource List**.

See also

- "Add multiple resources" on page 199
- "Start the Monitor" on page 185
- "Introduction to MobiLink users" [*MobiLink - Client Administration*]

Add multiple resources

As an administrator, you can add multiple resources to the Monitor by creating a comma separated values (CSV) file, and then importing the file.

To add multiple resources to the Monitor

1. Create a CSV file.

   Each line in the CSV file contains information about a single resource. Each comma-separated term within a line describes an attribute of the resource. The order of the terms is important. The following tables describe the terms and their order:

   - **CSV file format for MobiLink resources**

Position in the line	Term name	Description
1	Resource type	Type **ml** to indicate that the resource is a MobiLink server. This term is required.
2	Resource name	Specify the resource name as it will appear in the Monitor. This term is required.
3	Username	Specify the username used to connect to the resource. This term is required.
4	Password	Specify the password used to connect to the resource. This term is required.
5	Host (MobiLink server)	Specify the hostname or IP address of the computer on which the server is running. You can use the name localhost to represent the current computer. This term is required.

Position in the line	Term name	Description
6	Port (MobiLink server)	Specify the port number on which the MobiLink server is running. The default port number for MobiLink is 2439. This term is optional. The default value is 0
7	Connection type (MobiLink server)	Specify the method to use to connect to the resource: Type one of the following: ○ **tcpip** This is the default. ○ **http** ○ **https** ○ **tls** This term is optional.

Position in the line	Term name	Description
8	Encryption type (MobiLink server)	Specify the method used to encrypt the connection. Type one of the following: ○ **N** No encryption. This is the default. ○ **E** ECC encryption ○ **R** RSA encryption ○ **F** RSA-FIPS encryption. This term is optional.
9	Connection parameters (MobiLink server)	Specify additional connection parameters to use when connecting to the database. List the connection parameters as a semicolon delimited list of option=value pairs. This term is optional.

MobiLink servers:

The following is an example of an import file that contains two MobiLink server resources.

```
ml,demo2,DBA,sql,localhost
ml,demo4,DBA,sql,localhost,0,tcpip,N
```

- **CSV file format for MobiLink server farm resources**

Position in the line	Term name	Description
1	Resource type	Type **mf** to indicate that the resource is a MobiLink server farm. This term is required.
2	Resource name	Specify the resource name as it will appear in the Monitor. This term is required.
3	MobiLink server name	Specify the MobiLink server resources that are to be included in the MobiLink server farm. List the MobiLink server resources as a comma delimited list. This term is required.

MobiLink server farm:

The following is an example of an import file that contains two MobiLink server resources and a MobiLink server farm resource.

```
ml,demo2,DBA,sql,localhost
ml,demo4,DBA,sql,localhost,0,tcpip,N
mf,collection_demo,demo2,demo4
```

2. Log in as an administrator to the Monitor.
3. Click **Administration**.
4. Click **Resources**.
5. Click **Import**.
6. Locate the import file and click **Open**.

   The resources from the import file are added to the Monitor and monitoring starts.

7. Click **Close**.
8. Click **Close**.
9. The imported resource is added to the **Resource List** in the **Overview** dashboard.

Stop monitoring resources

You stop monitoring resources when you do not want the Monitor to collect metrics from a MobiLink server or a MobiLink server farm. For example, you want to stop monitoring when you know that the resource will be unavailable; otherwise, you receive alerts until the resource is available. Except for the default Monitor resource, you can stop monitoring any resource at any time.

When you stop monitoring a resource, the Monitor:

- Stops collecting metrics for the resource.
- Stops issuing alerts for the resource.

There are two ways to stop monitoring a resource:

- **Schedule a regular, repeating, blackout period** This method is a good choice when the following conditions apply:
  - You must repeatedly stop monitoring the MobiLink server. For example, you perform regular maintenance at the end of each month.
  - You know in advance how long the MobiLink server is unavailable. For example, you know that your regular maintenance takes four hours.
  - You need monitoring to automatically restart. When a blackout completes, the Monitor attempts to reconnect to the resource and to continue collecting data.

  To use this method, you create blackouts to make the Monitor stop monitoring at specified times. See "Automatically stop monitoring resources using blackouts" on page 205.
- **Manually stop the monitoring** This method is a good choice when the following conditions are met:
  - You need to stop monitoring for infrequent or one-time tasks. For example, you need to stop monitoring because the computer that the resource is running on needs to be taken off-line for special maintenance.
  - You are available to restart the monitoring afterward. When a resource has been stopped manually, the Monitor waits for you to restart the monitoring.

  To use this method, see "Manually stop monitoring resources" on page 205.

If you want to permanently stop monitoring a resource, you can remove it from the Monitor. See "Remove resources" on page 206.

Manually stop monitoring resources

The following procedure describes how to manually stop a resource. For information about what happens when you stop a resource, see "Stop monitoring resources" on page 204.

To manually stop a resource

1. Log in as an administrator to the Monitor.
2. Choose **Tools** » **Administration**.
3. Click **Resources**.
4. Select the resource, and then click **Stop**.

   Note
   When you stop monitoring a MobiLink server farm, monitoring stops for each individual MobiLink server that is part of the farm.

5. Click **Close**.

See also

- "Add resources" on page 197
- "Automatically stop monitoring resources using blackouts" on page 205

Automatically stop monitoring resources using blackouts

Blackouts are times when you do not want the Monitor to collect metrics. When a blackout completes, the Monitor attempts to reconnect to the resources and to continue collecting data.

The following procedure describes how to stop a resource using blackouts. For information about what happens when you stop a resource and about when you should use blackouts, see "Stop monitoring resources" on page 204.

Blackouts occur in the local time of the resource. See "Understanding how time is displayed" on page 195.

To configure the blackout time

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Click **Resources**.
4. Select the resource, and then click **Configure**.
5. Click **Blackouts**.

6. Click **New**.
7. In the **New Blackout Period** window, specify the date and time (24 hour clock) for the blackout.

   The time is local to the computer where the resource MobiLink server resides.
8. Click **Save**.
9. Click **Save**.
10. Click **OK**.
11. Click **Close**.

See also

- "Add resources" on page 197
- "Manually stop monitoring resources" on page 205

Remove resources

You should only remove resources when you are certain that you don't need to monitor them; for example, if the MobiLink server or the MobiLink server farm is no longer being used.

Removing a resource causes the Monitor to:

- Permanently stop monitoring the resource.
- Discard the metrics collected for the resource.

Only administrators can remove resources. You cannot delete the **SQL Anywhere Monitor** resource.

To remove a resource

1. Log in to the Monitor as an administrator.
2. Click **Administration**.
3. Click **Resources**.
4. Select the resource.
5. Click **Remove**.
6. Click **Yes**.

> **Note**
> When you remove a MobiLink server farm, the Monitor prompts you to remove the MobiLink servers that is part of the farm. Click **Yes** to remove the identified MobiLink servers or click **No** remove only the MobiLink server farm and not the identified MobiLink servers.

7. Click **Close**.

See also

- "Stop monitoring resources" on page 204

Metrics

The Monitor collects and stores metrics from MobiLink servers and MobiLink server farms, including, but not limited to:

- Whether the resource is running.
- Whether the computer that the resource is running on is running properly and is connected to the network.
- Whether the resource is listening and processing requests.
- The number of synchronizations that the MobiLink server performs over a period of time.

Collection intervals

Metrics displayed in the Monitor are only as precise as the collection interval. As an administrator, you can change the rate at which metrics are collected by the Monitor. By default, metrics are collected every 30 seconds.

To set the collection interval for a resource

1. Log in to the Monitor as an administrator.
2. Click **Resources**.
3. Select the resource, and then click **Configure**.
4. Click **Collection Interval**.
5. Specify the interval at which metrics are collected (hours:minutes:seconds). The collection interval must be at least 10 seconds long.
6. Click **Save**.
7. Click **OK**.
8. Click **Close**.

See also

- "Refresh metrics" on page 208

Export metrics

You can export metrics that have a graph or table associated with them, to an XML file. For example, most of the metrics in the **Key Performance Metrics** widget can be exported.

To export metrics

1. Log in to the Monitor.
2. Open a dashboard.
3. On a widget that displays the metrics, click the dropdown menu arrow, and then click **Export**.
4. Follow the instructions in the **Export Metrics** window to export metrics.
5. Click **Export**.
6. When prompted, specify a file name.

   An XML file is created containing the specified metrics.

Refresh metrics

By default, the Monitor display is automatically refreshed every minute. You can change the refresh the interval by clicking **Tools** » **User Settings**. This setting is independent of the collection interval rate for a resource, which specifies how often the Monitor collects metrics from the resource being monitored. See "Collection intervals" on page 207.

To set the refresh rate

1. Choose **Tools** » **User Settings**.
2. Change the settings as required. The default is one minute.
3. Click **OK**.

When you click **Refresh Data** the Monitor retrieves and displays the latest metrics.

To refresh metrics

- Click **Refresh Data**.

When you press F5, the Monitor reloads the browser, retrieves the metrics that the Monitor has collected to date, and displays the metrics.

To reload the Monitor

- Press F5.

List of metrics

List of metrics for MobiLink server resources

Metric name	Description
Authenticating Synchronizations	Shows the number of requests in the server that are currently in the authenticate user phase. For a description of the synchronization phases, see "-v mlsrv12 option" on page 70.
Bytes Read Rate	Shows the total number of bytes ever read from the network. See "TCP_BYTES_READ" on page 59.
Bytes Written Rate	Shows the total number of bytes ever written to the network. See "TCP_BYTES_WRITTEN" on page 60.
Cached Remote Tasks Requests	Shows the number of cached remote tasks requests. See "SIRT_NUM_REQUESTS" on page 59.
Command Processor Stage Length	Shows the length of the queue for synchronization work. See "CMD_PROCESSOR_STAGE_LEN" on page 57.
Commit Rate	Shows the rate at which commits occur.
Commits	Shows total number of commits. See "NUM_COMMITS" on page 57.
Completed Synchronization Rate	Shows the rate of successfully completed synchronizations for the server.
Connected Clients	Shows the number of connected clients. See "ML_NUM_CONNECTED_CLIENTS" on page 57.
Connections Closed Rate	Shows the rate that TCP connections are closed.

Metric name	Description
Connections Opened Rate	Shows the rate that TCP connections are opened.
Connections Rejected Rate	Shows the rate that TCP connections are rejected by the server.
Consolidated Database Type	Shows the type of consolidated database. For example, **SQL Anywhere**.
Consolidated Database Version	Shows the version of the consolidated database.
CPU Total Time	Shows the amount of CPU time used by the MobiLink server in microseconds. See "CPU_USAGE" on page 57.
CPU Usage	Shows the percentage of CPU time used by the MobiLink server.
CPUs	Shows the number of CPUs in use.
Current TCP Connections	Shows the number of TCP connections currently in use by this MobiLink server.
Database Connections In Use	Shows the number of database connections in use. See "DB_CONNECTIONS" on page 57.
Database Worker Thread Count	Shows the number of database worker threads. See "-w mlsrv12 option" on page 74.
Downloading Synchronizations	Shows the current number of synchronizations that are in the Prepare for Download, Fetch Download, Wait for Download Ack, or End Sync phases. For a description of the synchronization phases, see "-v mlsrv12 option" on page 70.

Metric name	Description
Driver Name	Shows the name of the driver for the consolidated database.
Driver Version	Shows the version of the driver for the consolidated database.
Errors	Shows the total number of errors. See "NUM_ERRORS" on page 57.
Failed Synchronization Rate	Shows the rate of failed synchronizations for the server, expressed in synchronizations per second.
Failed Synchronizations	Shows total number of failed synchronizations. See "NUM_FAILED_SYNCS" on page 57.
File Transfers	Shows the number of mlfiletransfers currently connected. See "NUM_CONNECTED_FILE_XFERS" on page 57.
Finished Synchronization Rate	Shows the rate at which synchronizations complete.
Free Disk Space For MobiLink Cache	Shows the disk space available on the temp disk for MobiLink cache in bytes. See "FREE_DISK_SPACE" on page 57.
Heartbeat Stage Length	Shows the length of the queue for periodic, non-synchronization work. See "HEARTBEAT_STAGE_LEN" on page 57.
Host	Shows the name of the computer running the MobiLink server. Typically, this is the computer's host name.
Is Consolidated Database Clustered?	Shows **True** if the MobiLink server is using a clustered consolidated database.
Is Primary Server Known?	Shows **Yes** if the primary server is known by the server. Shows **No** if the primary server is unknown or if the server does not care what the primary server is. See "PRIMARY_IS_KNOWN" on page 59.

Metric name	Description
Is Primary Server?	Shows **Yes** when the server is the primary server. See "SERVER_IS_PRIMARY" on page 59.
Last Checked Time	Shows the last time the Monitor collected data for the resource.
Licensed Company	Shows the name of the licensed company.
Licensed User	Shows the name of the licensed user.
Listeners	Shows the number of listeners currently connected. See "NUM_CONNECTED_LISTENERS" on page 57.
Max Cache Size	Shows the maximum cache size used by the MobiLink server
Max Clients	Shows the maximum number of clients. See "-sm mlsrv12 option" on page 67.
Max Database Connections	Shows the maximum number of database connections, as set by the -cn option or the -w option for mlsrv12. See "-cn mlsrv12 option" on page 43 and "-w mlsrv12 option" on page 74.
Max Synchronization Time	Shows the maximum time it took for a synchronization to occur. The age of the oldest synchronization in microseconds. See "LONGEST_SYNC" on page 57.
Max TCP Connections	Shows the maximum number of TCP connections, as set by the -nc option for mlsrv12. See "-nc mlsrv12 option" on page 52.
Max Uploads Allowed	Shows the maximum number of concurrent uploads to the database. See "-wu mlsrv12 option" on page 75.
Max Wait For Connection	Shows the longest length of time an active synchronization has been waiting for the database. See "LONGEST_DB_WAIT" on page 57.
Memory Used	Shows the bytes of RAM in use. Shows the Tracked Memory value when the server runs on a non-Windows platform. See "MEMORY_USED" on page 57 and "TRACKED_MEMORY" on page 60.

Metric name	Description
Monitors	Shows the number of SQL Anywhere Monitors and MobiLink Monitors currently connected. See "NUM_CONNECTED_MONITORS" on page 57.
Notifier Stage Length	Shows the length of the notifier work queue. See "NOTIFIER_STAGE_LEN" on page 57.
Operating System	Shows the operating system on which the software is running.
Outbound Enabler Stage Length	Shows the length of the integrated Outbound Enabler work queue. See "OE_STAGE_LEN" on page 59.
Pages In Stream Stack	Shows the number of pages held by the network streams. See "PAGES_IN_STREAMSTACK" on page 59.
Pages Locked	Shows the number of cache pages loaded into memory. See "PAGES_LOCKED" on page 59.
Pages Swapped In	Shows the total number of pages ever read from disk. See "PAGES_SWAPPED_IN" on page 59.
Pages Swapped In Rate	Shows rate at which the pages are read from disk. See "PAGES_SWAPPED_IN" on page 59.
Pages Swapped Out	Shows the total number of pages ever swapped to disk. See "PAGES_SWAPPED_OUT" on page 59.
Pages Swapped Out Rate	shows the rate at which the pages are swapped to disk. See "PAGES_SWAPPED_OUT" on page 59.
Pages Used	The number of cache pages used. This includes pages swapped to disk so it may be larger than the cache size. See "PAGES_USED" on page 59.
Percent of Clients Connected	Shows the percentage of clients that are currently connected. This metric is derived from dividing the **Number Of Connected Clients** value by the **Maximum Clients** value and multiplying the result by 100.

Metric name	Description
Percent of Connections In Use	Shows the percentage of connections that are currently in use.
Percent of Pages in Stream Stack	Shows the percentage of pages that are held by the network streams.
Percent of Pages Locked	Shows the percentage of pages that are locked.
Percent of Pages Used	Shows the percentage of pages that are in use. This includes pages swapped to disk so it may be larger than the cache size. See "PAGES_USED" on page 59.
Pings	Shows the number of pinging clients currently connected. See "NUM_CONNECTED_PINGS" on page 57.
Processor Architecture	Shows a string that identifies the processor type.
Raw TCP Stage Length	Shows the length of the network work queue. See "RAW_TCP_STAGE_LEN" on page 59.
Requests	Shows the number of valid requests being serviced.
Requests in Apply Upload Phase	Shows the number of synchronizations currently in the apply upload phase. See "mlsrv12 -vm option" on page 71.
Requests in Authenticate User Phase	Shows the number of synchronizations currently in the authenticate user phase. See "mlsrv12 -vm option" on page 71.

Metric name	Description
Requests in Begin Synchronization Phase	Shows the number of synchronizations currently in the begin synchronization phase. See "mlsrv12 -vm option" on page 71.
Requests in Connect for Download Ack Phase	Shows the number of synchronizations currently in the connect for download ack phase. See "mlsrv12 -vm option" on page 71.
Requests in Connect Phase	Shows the number of synchronizations currently in the connect phase. See "mlsrv12 -vm option" on page 71.
Requests in End Synchronization Phase	Shows the number of synchronizations currently in the end synchronization phase. See "mlsrv12 -vm option" on page 71.
Requests in Fetch Download Phase	Shows the number of synchronizations currently in the fetch download phase. See "mlsrv12 -vm option" on page 71.
Requests in Get DB Worker For Download Ack Phase	Shows the number of synchronizations currently in the get DB worker for ack phase. See "mlsrv12 -vm option" on page 71.
Requests in Get DB Worker Phase	Shows the number of synchronizations currently in the get DB worker phase. See "mlsrv12 -vm option" on page 71.
Requests in Non-Blocking Download Ack Phase	Shows the number of synchronizations currently in the non-blocking download ack phase. See "mlsrv12 -vm option" on page 71.

Metric name	Description
Requests in Prepare For Download Phase	Shows the number of synchronizations currently in the prepare for download phase. See "mlsrv12 -vm option" on page 71.
Requests in Receive Upload Phase	Shows the number of synchronizations currently in the receive upload phase. See "mlsrv12 -vm option" on page 71.
Requests in Send Download Phase	Shows the number of synchronizations currently in the send download phase. See "mlsrv12 -vm option" on page 71.
Requests in Synchronization Request Phase	Shows the number of synchronizations currently in the synchronization request phase. See "mlsrv12 -vm option" on page 71.
Requests in Wait For Download Ack Phase	Shows the number of synchronizations currently in the wait for download ack phase. See "mlsrv12 -vm option" on page 71.
Rollback Rate	Shows the rate at which rollbacks occur. See "NUM_ROLLBACKS" on page 58.
Rollbacks	Shows the total number of rollbacks. See "NUM_ROLLBACKS" on page 58.
Rows Downloaded	Shows the total number of rows sent to remotes. See "NUM_ROWS_DOWNLOADED" on page 58.
Rows Downloaded Rate	Shows rate at which rows are sent to remotes. See "NUM_ROWS_DOWNLOADED" on page 58.
Rows Uploaded	Shows the total number of rows received from remotes. See "NUM_ROWS_UPLOADED" on page 58.

Metric name	Description
Rows Uploaded Rate	Shows the rate at which rows are received from remotes. See "NUM_ROWS_UPLOADED" on page 58.
RTNotifier Light Weight Poll Hit Rate	Shows the rate of the number of lightweight polls from remote tasks agents that found at least one remote tasks request.
RTNotifier Light Weight Poll Hits	Shows the number of lightweight polls from remote task agents that found at least one remote task request. See "SIRT_NUM_LWP_HITS" on page 59.
RTNotifier Light Weight Poll Miss Rate	Shows the rate of the number of lightweight polls from remote tasks agents that found no remote tasks requests.
RTNotifier Light Weight Poll Misses	Shows the number of light weight polls from remote task agents that found no remote tasks. See "SIRT_NUM_LWPS" on page 59.
RTNotifier Light Weight Poll Rate	Shows the rate of the number of lightweight polls from remote task agents.
RTNotifier Light Weight Polls	Shows the number of lightweight polls from remote task agents. See "SIRT_NUM_REQUESTS" on page 59.
Server Cache Size	Shows the size of the server cache.
Server Name	Shows the name of the MobiLink server as specified by the -zs option for the connected server. The default value is **<default>**. See "-zs mlsrv12 option" on page 82.
Server Time Offset	Shows the time difference between the time on the computer that the Monitor is running and the time on the computer that you are using to view the Monitor data. See "Understanding how time is displayed" on page 195.

Metric name	Description
Server Tracked Memory Usage	Shows the amount of memory allocated by the server. Use this metric for non-Windows systems where the MEMORY_USED metric is unavailable. On Microsoft Windows systems, use the MEMORY_USED metric for increased accuracy. See "TRACKED_MEMORY" on page 60.
Start Time	Shows the time when the MobiLink server started.
Started Synchronization Rate	Shows the rate at which synchronizations are started.
Steam Stage Length	Shows the length of the high level network processing queue. See "STREAM_STAGE_LEN" on page 59.
Successful Synchronizations	Shows the total number of successful synchronizations. See "NUM_SUCCESS_SYNCS" on page 58.
Synchronization Error Rate	Shows the rate of synchronization errors.
Synchronization Warning Rate	Shows the rate of synchronization warnings for the server.
Synchronizations	Shows the number of data synchronizations currently connected. See NUM_CONNECTED_SYNCS.
Synchronizations Blocked	Shows the number of synchronizations that are currently waiting for a database connection.
Synchronizations Finished	Shows the total number of synchronizations that have finished.
Synchronizations Started	Shows the total number of synchronizations that have started.

Metric name	Description
TCP Bytes Read	Shows the total number of bytes ever read. See "TCP_BYTES_READ" on page 59.
TCP Bytes Written	Shows the total number of bytes ever written. See "TCP_BYTES_WRITTEN" on page 60.
TCP Connections Closed	Shows total number of connections ever closed. See "TCP_CONNECTIONS_CLOSED" on page 60.
TCP Connections Opened	Shows the total number of connections ever opened. See "TCP_CONNECTIONS_OPENED" on page 60.
TCP Connections Rejected	Shows the total number of connections ever rejected. "TCP_CONNECTIONS_REJECTED" on page 60.
Timed Work Stage Length	Shows the length of the dynamic caching work queue. See "TIMED_WORK_STAGE_LEN" on page 60.
Unavailable Since	Shows the time since the resource became unavailable.
Unknown Connected Clients	Shows the number of connected clients whose origins are unknown.
Unsubmitted Error Reports	Shows the number of unsubmitted error reports for the server. An error report is submitted when SQL Anywhere 12 software crashes. See "NUM_UNSUBMITTED_ERROR_RPTS" on page 59 and "Suppress alerts for unsubmitted error reports from resources" on page 237.
Upload Connections In Use	Shows the number of upload connections currently in use. See "NUM_UPLOAD_CONNS_IN_USE" on page 59.
Uploading Synchronization	Shows the number of synchronizations that are in the upload phase.
Version	Shows the version of the software being run.

Metric name	Description
VM Memory Usage	Shows the amount of memory used by any attached VMs. See "VM_MEM_USE" on page 60.
Waiting For Database Connection	Number of requests that are waiting for a database connection.
Warnings	Shows the total number of warnings. See "NUM_WARNINGS" on page 59.

List of metrics for MobiLink server farm resources

Metric name	Scope	Description
Authenticating Synchronizations	Average	Shows the number of requests in the server that are currently in the authenticate user phase. For a description of the synchronization phases, see "-v mlsrv12 option" on page 70.
Bytes Read Rate	Average	Shows the total number of bytes ever read from the network. See "TCP_BYTES_READ" on page 59.
Bytes Written Rate	Average	Shows the total number of bytes ever written to the network. See "TCP_BYTES_WRITTEN" on page 60.
Cached Remote Task Requests	Average	Shows the number of cached remote tasks requests. See "SIRT_NUM_REQUESTS" on page 59.
Command Processor Stage Length	Average	Shows the length of the queue for synchronization work. See "CMD_PROCESSOR_STAGE_LEN" on page 57.
Commit Rate	Average	Shows the rate at which commits occur.
Commits	Average	Shows total number of commits. See "NUM_COMMITS" on page 57.

Metric name	Scope	Description
Completed Synchronization Rate	Total	Shows the rate of successfully completed synchronizations for the server.
Connected Clients	Average	Shows the number of connected clients. See "ML_NUM_CONNECTED_CLIENTS" on page 57.
Connections Closed Rate	Average	Shows the rate that TCP connections are closed.
Connections Opened Rate	Average	Shows the rate that TCP connections are opened.
Connections Rejected Rate	Average	Shows the rate that connections are rejected by the server.
CPU Total Time	Total	Shows the amount of CPU time used by the MobiLink server in microseconds. See "CPU_USAGE" on page 57.
CPU Usage	Average	Shows the average amount of CPU used by the MobiLink server.
Current TCP Connections	Total	Shows the number of TCP connections currently in use by this MobiLink server.
Database Connections In Use	Total	Shows the number of database connections in use. See "DB_CONNECTIONS" on page 57.
Downloading Synchronizations	Average	Shows the current number of synchronizations that are in the Prepare for Download, Fetch Download, Wait for Download Ack, or End Sync phases. For a description of the synchronization phases, see "-v mlsrv12 option" on page 70.

Metric name	Scope	Description
Errors	Average	Shows the average number of errors. See “NUM_ERRORS” on page 57.
Failed Synchronization Rate	Total	Shows the rate of failed synchronizations for the server.
Failed Synchronizations	Average	Shows total number of failed synchronizations. See “NUM_FAILED_SYNCS” on page 57.
File Transfers	Total	Shows the number of mlfiletransfers currently connected. See “NUM_CONNECTED_FILE_XFERS” on page 57.
Finished Synchronization Rate	Total	Shows the rate at which synchronizations complete.
Free Disk Space For MobiLink Cache	Average	Shows the disk space available on the temp disk for MobiLink cache in bytes. See “FREE_DISK_SPACE” on page 57.
Heartbeat Stage Length	Average	Shows the length of the queue for periodic, non-synchronization work. See “HEARTBEAT_STAGE_LEN” on page 57.
Listeners	Total	Shows the number of listeners currently connected. See “NUM_CONNECTED_LISTENERS” on page 57.
Max Synchronization Time	Maximum	Shows the maximum time it took for a synchronization to occur. The age of the oldest synchronization in microseconds. See “LONGEST_SYNC” on page 57.
Max Wait For Connection	Maximum	Shows the longest length of time an active synchronization has been waiting for the database. See “LONGEST_DB_WAIT” on page 57.

Metric name	Scope	Description
Memory Used	Average	Shows the bytes of RAM in use. Shows the Tracked Memory value when the server runs on a non-Windows platform. See "MEMORY_USED" on page 57 and "TRACKED_MEMORY" on page 60.
Monitors	Total	Shows the number of SQL Anywhere Monitors and MobiLink Monitors currently connected. See "NUM_CONNECTED_MONITORS" on page 57.
Notifier Stage Length	Average	Shows the length of the notifier work queue. See "NOTIFIER_STAGE_LEN" on page 57.
Outbound Enabler Stage Length	Average	Shows the length of the integrated Outbound Enabler work queue. See "OE_STAGE_LEN" on page 59.
Pages in Stream Stack	Average	Shows the number of pages held by the network streams. See "PAGES_IN_STREAMSTACK" on page 59.
Pages Locked	Average	Shows the number of cache pages loaded into memory. See "PAGES_LOCKED" on page 59.
Pages Swapped In	Average	Shows the total number of pages ever read from disk. See "PAGES_SWAPPED_IN" on page 59.
Pages Swapped In Rate	Average	Shows rate at which the pages are read from disk. See "PAGES_SWAPPED_IN" on page 59.
Pages Swapped Out	Average	Shows the total number of pages ever swapped to disk. See "PAGES_SWAPPED_OUT" on page 59.
Pages Swapped Out Rate	Average	Shows the rate at which the pages are swapped to disk. See "PAGES_SWAPPED_OUT" on page 59.
Pages Used	Average	The number of cache pages used. This includes pages swapped to disk, so it may be larger than the cache size. See "PAGES_USED" on page 59.

Metric name	Scope	Description
Percent of Clients Connected	Average	Shows the percentage of clients that are currently connected. This metric is derived from dividing the **Number Of Connected Clients** value by the **Maximum Clients** value and multiplying the result by 100.
Percent of Connections In Use	Average	Shows the percentage of connections that are currently in use.
Percent of Pages in Stream Stack	Average	Shows the percentage of pages that are held by the network streams.
Percent of Pages Locked	Average	Shows the percentage of pages that are locked.
Percent of Pages Used	Average	Shows the percentage of pages that are in use. This includes pages swapped to disk so it may be larger than the cache size. See PAGES_USED.
Pings	Total	Shows the number of pinging clients currently connected. See "NUM_CONNECTED_PINGS" on page 57.
Raw TCP Stage Length	Average	Shows the length of the network work queue. See "RAW_TCP_STAGE_LEN" on page 59.
Requests	Total	Shows the number of valid requests being serviced.
Requests in Apply Upload Phase	Average	Shows the number of synchronizations currently in the apply upload phase. See "mlsrv12 -vm option" on page 71.
Requests in Authenticate User Phase	Average	Shows the number of synchronizations currently in the authenticate user phase. See "mlsrv12 -vm option" on page 71.

Metric name	Scope	Description
Requests in Begin Synchronization Phase	Average	Shows the number of synchronizations currently in the begin synchronization phase. See “mlsrv12 -vm option” on page 71.
Requests in Connect for Download Ack Phase	Average	Shows the number of synchronizations currently in the connect for download ack phase. See “mlsrv12 -vm option” on page 71.
Requests in Connect Phase	Average	Shows the number of synchronizations currently in the connect phase. See “mlsrv12 -vm option” on page 71.
Requests in End Synchronization Phase	Average	Shows the number of synchronizations currently in the end synchronization phase. See “mlsrv12 -vm option” on page 71.
Requests in Fetch Download Phase	Average	Shows the number of synchronizations currently in the fetch download phase. See “mlsrv12 -vm option” on page 71.
Requests in Get DB Worker For Download Ack Phase	Average	Shows the number of synchronizations currently in the get DB worker for ack phase. See “NUM_IN_GET_DB_WORKER_FOR_ACK” on page 58.
Requests in Get DB Worker Phase	Average	Shows the number of synchronizations currently in the get DB worker phase. See “mlsrv12 -vm option” on page 71.

Metric name	Scope	Description
Requests in Non-Blocking Download Ack Phase	Average	Shows the number of synchronizations currently in the non-blocking download ack phase. See "mlsrv12 -vm option" on page 71.
Requests in Prepare For Download Phase	Average	Shows the number of synchronizations currently in the prepare for download phase. See "mlsrv12 -vm option" on page 71.
Requests in Receive Upload Phase	Average	Shows the number of synchronizations currently in the receive upload phase. See "mlsrv12 -vm option" on page 71.
Requests in Send Download Phase	Average	Shows the number of synchronizations currently in the send download phase. See "mlsrv12 -vm option" on page 71.
Requests in Synchronization Request Phase	Average	Shows the number of synchronizations currently in the synchronization request phase. See "mlsrv12 -vm option" on page 71.
Requests in Wait For Download Ack Phase	Average	Shows the number of synchronizations currently in the wait for download ack phase. See "mlsrv12 -vm option" on page 71.
Rollback Rate	Average	Shows the total number of rollbacks. See "NUM_ROLLBACKS" on page 58.
Rollbacks	Average	Shows the total number of rollbacks. See "NUM_ROLLBACKS" on page 58.

Metric name	Scope	Description
Rows Downloaded	Average	Shows the total number of rows sent to remotes. See "NUM_ROWS_DOWNLOADED" on page 58.
Rows Downloaded Rate	Average	Shows the total number of rows sent to remotes. See "NUM_ROWS_DOWNLOADED" on page 58.
Rows Uploaded	Average	Shows the total number of rows received from remotes. See "NUM_ROWS_UPLOADED" on page 58.
Rows Uploaded Rate	Average	Shows the total number of rows received from remotes. See "NUM_ROWS_UPLOADED" on page 58.
RTNotifier Light Weight Poll Hit Rate	Average	Shows the rate of the number of lightweight polls from remote tasks agents that found at least one remote tasks request.
RTNotifier Light Weight Poll Hits	Average	Shows the number of lightweight polls from remote task agents that found at least one remote task request. See "SIRT_NUM_LWP_HITS" on page 59.
RTNotifier Light Weight Poll Miss Rate	Average	Shows the rate of the number of lightweight polls from remote tasks agents that found no remote tasks requests.
RTNotifier Light Weight Poll Misses	Average	Shows the number of light weight polls from remote task agents that found no remote tasks See "SIRT_NUM_LWPS" on page 59.
RTNotifier Light Weight Poll Rate	Average	Shows the rate of the number of lightweight polls from remote task agents.

Metric name	Scope	Description
RTNotifier Light Weight Polls	Average	Shows the number of lightweight polls from remote task agents. See "SIRT_NUM_REQUESTS" on page 59.
Server Cache Size	Average	Shows the size of the server cache.
Server Tracked Memory Usage	Average	Shows the amount of memory allocated by the server. Use this metric for non-Windows systems where the MEMORY_USED metric is unavailable. On Microsoft Windows systems, use the MEMORY_USED metric for increased accuracy. See "TRACKED_MEMORY" on page 60.
Started Synchronization Rate	Total	Shows the rate at which synchronizations are started.
Stream Stage Length	Average	Shows the length of the high level network processing queue. See "STREAM_STAGE_LEN" on page 59.
Successful Synchronizations	Average	Shows the total number of successful synchronizations. See "NUM_SUCCESS_SYNCS" on page 58.
Synchronization Error Rate	Total	Shows the rate of synchronization errors.
Synchronization Warning Rate	Total	Shows the rate of synchronization warnings for the server.
Synchronizations	Total	Shows the number of data synchronizations currently connected. See NUM_CONNECTED_SYNCS "NUM_CONNECTED_SYNCS" on page 57.
Synchronizations Blocked	Average	Shows the number of synchronizations that are currently waiting for a database connection.

Metric name	Scope	Description
Synchronizations Finished	Total	Shows the number of synchronizations that have been completed.
Synchronizations Started	Total	Shows the total number of synchronizations that have started.
TCP Bytes Read	Average	Shows the total number of bytes ever read. See "TCP_BYTES_READ" on page 59.
TCP Bytes Written	Average	Shows the total number of bytes ever written. See "TCP_BYTES_WRITTEN" on page 60.
TCP Connections Closed	Average	Shows total number of connections ever closed. See "TCP_CONNECTIONS_CLOSED" on page 60.
TCP Connections Opened	Average	Shows the total number of connections ever opened. See "TCP_CONNECTIONS_OPENED" on page 60.
TCP Connections Rejected	Average	Shows the total number of connections ever rejected. "TCP_CONNECTIONS_REJECTED" on page 60.
Timed Work Stage Length	Average	Shows the length of the dynamic caching work queue. See "TIMED_WORK_STAGE_LEN" on page 60.
Unknown Connected Clients	Average	Shows the number of connected clients whose origins are unknown.
Upload Connections In Use	Average	Shows the number of upload connections currently in use. See "NUM_UPLOAD_CONNS_IN_USE" on page 59.

Metric name	Scope	Description
Uploading Synchronization	Average	Shows the number of synchronizations that are in the upload phase.
VM Memory Usage	Average	Shows the amount of memory used by any attached VMs. "VM_MEM_USE" on page 60.
Waiting for Database Connection	Average	Number of requests that are waiting for a database connection.
Warnings	Average	Shows the total number of warnings. See "NUM_WARNINGS" on page 59.

Monitor users

You must log in to the Monitor with a user name and password. The user name and password for logging in to the Monitor are case sensitive.

Default user

By default, when you first start the Monitor, it has one administrator user, named **admin**, with the password **admin**. By default, this user has full permissions. It is recommended that you change the default administrator password to restrict access to the Monitor. See "Edit Monitor users" on page 233.

The Monitor supports three types of users:

User type	Description
Read-only user	Has read-only access to monitor resources. Read-only users can view the metrics, but cannot access the **Administration** window. When you create a user from the log-in screen, this user is a Read-only user. A user name and password are required.
Operator	Has read-only access to monitor resources and can receive alerts. These users can view the metrics, can receive email alerts, and can resolve and delete alerts. However, operators cannot access the **Administration** window. A user name and password are required.

User type	Description
Administrator	Has the same access as an operator, and can also configure resources and add users. Administrators can also access the **Administration** window. The default user, **admin**, is an administrator. A user name and password are required.

All users, once they have logged in, can check their user type and change their settings. However, only an administrator can change a user's type. See "Edit Monitor users" on page 233.

To check your Monitor user type

1. Log in to the Monitor.
2. Choose **Tools** » **User Settings**.
3. Review the **User Type** setting in the window.

See also

- "Administration window" on page 195

Create Monitor users

By default, from the log-in screen, anyone can create their own read-only user and have read-only access to the Monitor. However, administrators can change this default so that only administrators can create users; see "Only allow administrators the ability to create users" on page 234.

To create a read-only user from the log-in screen

1. Click **Create New User**.

   Note
   If the **Create New User** link is not available, then the administrator has changed the default behavior so that only administrators can create new users. Contact your Monitor administrator to create a new user.

2. Fill in the information for the new user.

   An email address is only required if you want to receive email alerts from the Monitor. Contact your Monitor administrator to change your user type to operator or administrator, and sign you up to receive email alerts. See "Send alert emails" on page 238.

When logged in, administrators can create any type of user, including other administrators.

To create new users when logged-in as an Administrator

1. Log in to the Monitor as an administrator.

2. Choose **Tools** » **Administration**.
3. Click **Users**.
4. Click **New**.
5. Fill in the information for the new user. An email address is only required for users who should receive email alerts from the Monitor.
6. Select a language from the **Preferred Language Type** dropdown menu. The specified language sets the language used by the Monitor, including the language used in alerts.
7. Select a user type. Each type has different permissions. See "Monitor users" on page 230.
8. Click **Next** to create the resource dashboards for this user.
9. Click **Save**.
10. Click **Close**.
11. If you created an operator or an administrator user, this user can receive alert notifications by email. See "Send alert emails" on page 238.

See also

- "Edit Monitor users" on page 233
- "Only allow administrators the ability to create users" on page 234

Associate Monitor users with resources

You must associate an operator or administrator user with a resource if you want the user to receive email alerts about the associated resource.

To associate an operator or administrator with a resource

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Ensure that the operator or administrator has an email address specified in their user account.

   Click **Users** and verify that the user has an email address specified.
4. Click **Resources**, select a resource from the list, and then click **Configure**.
5. Click **Operators**.
6. Select a user from the **Available Operators** list and then click **Add**.

   The user name appears in the **Selected Operators** list.

7. Click **Save**.
8. Click **OK**.
9. Click **Close**.
10. Ensure that the Monitor is set up to send alert notifications by email. See "Enable the Monitor to send alert emails" on page 239.

See also

- "Send alert emails" on page 238

Edit Monitor users

All users, once they have logged in, can change their user settings by clicking **Tools** » **User Settings**. However, only an administrator can change a user's type.

To edit an existing Monitor user

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Click **Users**.
4. Select the user to edit, and then click **Edit**.
5. Change the settings for the user as required. An email address is only required for users who should receive email alerts from the Monitor.
6. Click **Save**.
7. Click **Close**.
8. If you created an operator or an administrator user, this user can receive alert notifications by email. See "Send alert emails" on page 238.

See also

- "Create Monitor users" on page 231
- "Delete Monitor users" on page 233

Delete Monitor users

Deleting a user removes the user from the Monitor and disassociates the user from any resource.

To delete an existing Monitor user

1. Log in to the Monitor as an administrator.

2. Choose **Tools** » **Administration**.
3. Click the **Users**.
4. Select the user, and then click **Delete**.
5. Click **Yes** to delete the selected user.

   The user is deleted from the Monitor.
6. Click **Close**.

See also

- "Create Monitor users" on page 231
- "Edit Monitor users" on page 233

Only allow administrators the ability to create users

By default, from the log-in screen, anyone can create their own user name and password and have read-only access to the Monitor. However, as an administrator, you turn off this feature so that only administrators can create users.

To restrict user creation to administrators

1. Log in to the Monitor as an administrator.
2. Chose **Tools** » **Administration**.
3. Click **Configuration**.
4. Click **Edit**.
5. Click **Options**.
6. Clear the **Allow Anyone Read-only Access To The SQL Anywhere Monitor** option.
7. Click **Save**.
8. Click **Close**.

See also

- "Create Monitor users" on page 231
- "Edit Monitor users" on page 233

Alerts

An **alert** is a condition or state of interest about a resource that should be brought to an administrator's or operator's attention. Alerts are detected by the Monitor based on metrics that are collected. They are not detected at the MobiLink serverthe Relay Server farm being monitored.

There are predefined alerts for conditions such as low disk space, failed login attempts, and high memory usage. You can change the default threshold values by editing the resource. See "Specify alert thresholds" on page 236.

When an alert condition is met, the alert is listed in the **Alert List** widget for the specified resource. See "Alerts List widget" on page 192.

By default, alerts appear in the **Alert List** widgets and they include information about the cause of the problem, and provide advice for resolving the problem. In the **Resource List** the resource's status changes to reflect the existence and severity of the alert. See "Overview dashboard" on page 190.

You can configure the Monitor to send an email to operators and administrators when an alert occurs. See "Send alert emails" on page 238.

View alerts

The Monitor keeps only the most recent 50 alerts in the alert list.

> **Note**
> Any logged-in user can view alerts; however, only operators and administrators can resolve and delete alerts.

To view an alert

1. Choose **Alerts** » **Today**.
2. Select an alert from the list, and then click **Details**.
3. Click **OK**.

See also

- "Resolve alerts" on page 235
- "Delete alerts" on page 236
- "Send alert emails" on page 238

Resolve alerts

As an operator or an administrator, you can mark an alert as resolved after the issue that triggered the alert has been addressed.

Resolving an alert causes the Monitor to change the alert **Status** to **Resolved by *user-name***, but leaves the alert in the alert list. If you want to remove the alert from the list, you must delete it. See "Delete alerts" on page 236.

To resolve an alert

1. Log in to the Monitor as an administrator or operator user.
2. Click **Overview**.
3. In the **Alerts List** widget, select an alert from the list and click **Mark Resolved** to resolve the selected alert.

   The value in the **Status** column changes to **Resolved by *your-user-name***.

   If this alert was the resource's only unresolved alert, the resource's status in the **Resource List** widget changes to Healthy (no icon is present).

See also

- “Overview dashboard” on page 190
- “Delete alerts” on page 236
- “Send alert emails” on page 238
- “View alerts” on page 235
- “Alerts” on page 234

Delete alerts

As an operator or an administrator, you can delete any alert from an **Alerts List**. You can delete alerts, regardless of their statuses.

To delete alerts

1. Log in to the Monitor as an administrator or operator user.
2. Click **Overview**.
3. In the **Alerts List** widget, select an alert from the list and click **Delete**.

   The alert is removed from the alerts list.

See also

- “Resolve alerts” on page 235
- “Send alert emails” on page 238
- “View alerts” on page 235
- “Alerts” on page 234

Specify alert thresholds

As an administrator you can configure when alerts should be issued. But, you cannot configure what metrics the Monitor collects nor can you configure the default resource, the SQL Anywhere Monitor.

To configure when alerts should be issued

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Select **Resources**, and then select a resource from the list.
4. Click **Configure**.
5. Click **Alert Thresholds**. Edit the thresholds. For definitions of the alerts, see "Alert thresholds" on page 238.
6. Configure the other settings as required.
7. Click **Save**.
8. Click **OK**.
9. Click **Close**.

See also

- "Metrics" on page 207
- "Alert thresholds" on page 238

Suppress alerts for unsubmitted error reports from resources

As an administrator, you can configure whether the Monitor sends out alerts when resources have unsubmitted error reports. By default, the Monitor does not send these alerts. For information about error reports and about how to submit them, see "Error reporting in SQL Anywhere" [*SQL Anywhere Server - Database Administration*].

To suppress alerts for unsubmitted error reports

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Click **Configuration**.
4. Click **Edit**.
5. Click **Options**.
6. Select **Suppress unsubmitted error report alerts form resources**.
7. Click **Save**.
8. Click **Close**.

Alert thresholds

As an administrator, you can configure the thresholds that are used to trigger alerts. See "Specify alert thresholds" on page 236.

The following list describes the alerts and their default thresholds.

Because each synchronization system has different behaviors and constraints, the defaults may be inappropriate for your environment. You should review each alert threshold and adjust them as necessary.

Alert thresholds for MobiLink server resources

- **Alert When CPU Usage Exceeds The Given Threshold For The Given Number Of Seconds** The **Threshold** default is 100 percent. The **Seconds** default is 300.
- **Alert When The Percentage Of Cache Pages Used Is Greater Than (%)** The default is 100.
- **Alert When The Percentage Of Locked Cache Pages Is Greater Than (%)** The default is 80.
- **Alert When The Number Of Pages Being Swapped In And Out Per Second Exceeds The Given Threshold For The Given Number Of Seconds** The threshold default is 256. The seconds default is 120.
- **Alert When Longest Active Synchronization Time Is Greater Than (Seconds)** The default is 600.
- **Alert When The Number Of Failed Synchronizations Exceeds The Given Threshold For The Given Number Of Minutes** The threshold of failed synchronizations default is 20. The minutes default is 60.
- **Alert When The Number Of Errors Exceeds The Given Threshold For The Given Number Of Minutes** The Threshold (Errors) default is 50. The Minutes default is 60.
- **Alert When Free Disk Space For MobiLink Cache Is Less Than (MB)** The default is 100.
- **Alert When The Longest Active Wait For Database Worker Thread Is Greater Than (Seconds)** The default is 300.
- **Suppress Alerts For The Same Condition That Occur Within Minutes** This option prevents you from receiving duplicate alerts within a specified time. The default is 30 minutes.

Send alert emails

As an administrator, you can configure the Monitor to send an email to specified operators and administrators when an alert occurs for specified resources.

To have the Monitor send alert notifications by email

1. Log in to the Monitor as an Administrator.

2. Create an administrator or operator with an email address. See "Create Monitor users" on page 231.
3. Associate the administrator or operator with a resource. See "Associate Monitor users with resources" on page 232.
4. Enable the Monitor to send emails. See "Enable the Monitor to send alert emails" on page 239.

Enable the Monitor to send alert emails

As an administrator, you can configure the Monitor to send emails to operators and administrators when alerts occur. The Monitor supports the SMTP and MAPI protocols for sending emails.

To enable the Monitor to send alert notifications by email

1. Log in to the Monitor as an Administrator.
2. Choose **Tools** » **Administration**.
3. Click **Configuration**.
4. Click **Edit**.
5. Click **Alert Notification**.
6. Select **Send Alert Notifications By Email**.
7. Choose either SMTP or MAPI for the **Which Protocol Do You Want To Use To Send Alerts By Email?** field.
8. Configure the other settings as required.
   - **MAPI**
     - **User Name** Type the user name for the MAPI server.
     - **Password** Type the password for the MAPI server.
   - **SMTP**
     - **Server** Specify which SMTP server to use. Type the server name or the IP address for the SMTP server. For example, *SMTP.yourcompany.com*.
     - **Port** Specify the port number to connect to on the SMTP server. The default is 25.
     - **Sender Name** Specify an alias for the sender's email address. For example, *JoeSmith*.
     - **Sender Address** Specify the email address of the sender. For example, *jsmith@emailaddress.com*.
     - **This SMTP Server Requires Authentication** Select this option if your SMTP server requires authentication.
       - **User Name** Specify the user name to provide to SMTP servers requiring authentication.
       - **Password** Specify the password to provide to SMTP servers requiring authentication.

9. Click **Send Test Email**.
10. When prompted, enter an email address to send the test email to and click **OK**.

    A test email is sent to the email address specified.
11. Click **Save**.
12. Click **Close**.

    When an alert occurs, the Monitor sends alert emails to operators and administrators who have specified email addresses in their user accounts. These users receive emails for the resources that they are associated with. See "Create Monitor users" on page 231 and "Associate Monitor users with resources" on page 232.

See also

- "Resolve alerts" on page 235
- "Delete alerts" on page 236
- "View alerts" on page 235

Back up the Monitor

By default, the Monitor performs maintenance on the metrics once a day at midnight. Maintenance affects metrics, not alerts.

As an administrator, you can:

- Schedule the Monitor to back up metrics.
- Control the amount of disk space that the Monitor uses.
- Perform maintenance on demand.

To back up metrics and control the amount of disk space used to store metrics

1. Log in to the Monitor as an Administrator.
2. Click **Administration**.
3. Select **Configuration**, and then click **Edit**.
4. Click **Maintenance**.
5. Specify a time (24-hour clock) when the Monitor should perform maintenance. By default, it performs maintenance at midnight. The time is local to the computer where the Monitor is running. See "Understanding how time is displayed" [*SQL Anywhere Server - Database Administration*].

6. Specify a directory where the Monitor should save the backed up data. The directory must exist on the computer where the Monitor is running.

7. Customize the **Data Reduction** settings:
   - **Reduce Metrics To A Representative Daily Value For Metrics Older Than** When you select this option, an average is taken for all numeric metrics that are older than the specified number of days, and then the numeric metrics are deleted. Non-numeric metrics are not deleted.
   - **Delete Values Older Than** When you select this option, all metrics that are older than the specified length of time are deleted.
   - **Delete Old Metrics When The Total Disk Space Used By The SQL Anywhere Monitor Becomes Greater Than (MB)** When you select this option, you specify the maximum amount of space that can be used to store the metrics. When the amount of disk space used reaches or exceeds the amount specified, the Monitor deletes metrics, starting with the oldest metrics. Metrics are deleted until enough free space exists to store new metrics.

8. Click **Perform Maintenance Now** to run the backup immediately. When prompted, click **OK**.

9. Click **Save**.

10. Click **Close**.

The following procedure describes how to replace the Monitor database with a backup copy.

To restore a backup copy of the Monitor database

1. Stop the Monitor, if it is running.

2. From your backup directory, copy the *samonitor.db* database file and *samonitor.log* log file.

3. Paste these files into the directory where the current Monitor database and log file are located. When prompted, overwrite the existing files.

   The default locations of the version 12.0.0 Monitor database files are listed in the following table:

Operating system	Monitor directory
Windows XP (installed with SQL Anywhere)	`C:\Documents and Settings\All Users\Documents\SQL Anywhere 12\Monitor\samonitor.db`
Windows XP (installed on a separate computer)	`C:\Documents and Settings\All Users\Documents\SQL Anywhere 12 Monitor\samonitor.db`
Windows Vista (installed with SQL Anywhere)	`C:\Users\Public\Documents\SQL Anywhere 12\Monitor \samonitor.db`
Windows Vista (installed on a separate computer)	`C:\Users\Public\Documents\SQL Anywhere 12 Monitor \samonitor.db`

Operating system	Monitor directory
Linux (installed with SQL Anywhere)	`/opt/sqlanywhere12/samonitor.db`
Linux (installed on a separate computer)	`/opt/sqlanywhere12/samonitor.db`

4. Re-start the Monitor.

Installing the SQL Anywhere Monitor in a production environment

It is recommended, particularly in production environments, that you:

1. Install and run the Monitor Production Edition.

   Advantages to running the Monitor Production Edition include:

   - The Monitor runs in the background as a service.
   - The Monitor starts automatically when the computer starts.
   - Upgrades and updates of SQL Anywhere do not overwrite or affect the Monitor Production Edition. In contrast, upgrades and updates of SQL Anywhere can affect the Monitor Developer Edition as it uses the installed SQL Anywhere on the back-end.

2. Install the Monitor on a computer that is different from the computer where the resources are running.

   Advantages to running the Monitor on a separate computer include:

   - The impact on the MobiLink server or other applications is minimized.
   - Monitoring is not affected if something happens to the computer where the resources are installed.

These instructions explain how to install the SQL Anywhere Monitor Production Edition.

To install the Monitor Production Edition (Windows)

1. Run the *setup.exe* file from the *Monitor* directory on your installation media, and follow the instructions provided.

   On Windows, the Monitor service is started by the installation..

2. Open the default URL for logging in to the Monitor: *http://localhost:4950*.

Note
If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 189.

3. Log in.

   When prompted, enter your user name and password for the Monitor. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 230.

To install the Monitor Production Edition (Linux)

1. As the root user, run the *setup.tar* file from the *Monitor* directory on your installation media, and follow the instructions provided.

   Note
   On Linux, the Monitor Production Edition can only be run by the root user.

2. On Linux, by default, the Monitor Production Edition automatically starts the Monitor service.

3. Open the default URL for logging in to the Monitor: *http://localhost:4950*.

   Note
   If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 189.

4. Log in.

   When prompted, enter your user name and password for the Monitor. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 230.

Upgrading the Monitor and migrating resources and metrics

Caution
Uninstalling the Monitor removes the application, as well as the resources and collected metrics.

If you want to preserve your current Monitor resources and metrics, you must:

1. Install a new version of the Monitor.
2. Migrate the resources and metrics.
3. Uninstall the older version of the Monitor.

See "Upgrading the SQL Anywhere Monitor and migrating resources and metrics" [*SQL Anywhere 12 - Changes and Upgrading*].

Securing the Monitor

You can secure communications between both the Monitor and your browser, and between the Monitor and resources it monitors.

Securing communications between the Monitor and your browser using transport-layer security (TLS)

You can use transport-layer security (TLS) to secure communication between the Monitor and your browser. The Monitor runs a web server that supports HTTPS connections using SSL version 3.0 and TLS version 1.0.

To set up TLS security for the Monitor

1. Obtain digital certificates from a certificate authority or create self-signed certificates with the Certificate Creation utility (createcert). See "Certificate Creation utility (createcert)" [*SQL Anywhere Server - Database Administration*].
2. Alter the Monitor start line string to use the certificates.
3. Configure your browser to accept your new certificates, if required.

For more information, see http://www.sybase.com/detail?id=1063938.

Securing connections between the Monitor and your resources

You can use ECC or FIPS to encrypt communications between the Monitor and the resources it monitors. ECC encryption and FIPS-certified encryption require a separate license.

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*].

Troubleshooting the Monitor

In addition to the recommendations listed below, Administrators can use the Message Log and Exception Reports features to troubleshoot the Monitor. See "Message Log" on page 196 and "Exception Reports" on page 197.

Problem	Recommendation
When you press F5 to refresh the browser window, you are required to log in to the Monitor.	Enable JavaScript in your browser.
You receive a network communication error when you try to log in to the Monitor.	Start the Monitor. See "Start the Monitor" on page 185.
After upgrading to the latest version of Adobe Flash Player you continue to receive instructions to upgrade Adobe Flash Player.	Verify that the installed version Adobe Flash Player is supported by your operating system. The Monitor is backwards compatible with version 10 of Adobe Flash Player. To determine the correct version, see http://www.adobe.com/products/flashplayer/systemreqs/.
The Monitor is unable to start monitoring a SQL Anywhere database resource.	Verify that the resource's password verification functions and login procedures allow the user sa_monitor_user to connect to the resource.
You are not receiving any alert emails.	Verify that the Monitor is properly configured to send emails and send a test email. See "Enable the Monitor to send alert emails" on page 239. Verify that the alert emails from the Monitor are not being blocked by a virus scanner. See "xp_startsmtp system procedure" [*SQL Anywhere Server - SQL Reference*].
The number of unscheduled requests reported by the Monitor appears to be less than the actual number of unscheduled requests.	When collecting metrics about the number of unscheduled requests, the Monitor executes query on the resource. This query could be an unscheduled request. Unscheduled queries are processed sequentially as they arrive. Therefore, if there are unscheduled requests when the Monitor attempts to execute its query, then this query must wait for the existing unscheduled requests to complete before it can execute. As a result, when the Monitor collects the number of unscheduled requests, this number does not include the unscheduled requests that existed between the time when the Monitor issued its query and the query executed.

Problem	Recommendation
You are not receiving alerts when the database disk space surpasses the specified threshold.	Between Monitor collection intervals, it is possible for a database to exceed the specified disk space alert threshold and the amount of space available. In such a case, the database would stop responding before the Monitor could collect the disk usage metrics and issue an alert. If your database grows quickly, set the disk space alert threshold to a higher number so that you can receive an alert before the database runs out of space. See "Alert thresholds" on page 238.
You can't see the **Administration** window when you are logged into the Monitor.	You must be logged in to the Monitor as an administrator to have access to the **Administration** window. See "Monitor users" on page 230.
You uninstalled the Monitor before migrating your data and now you have lost your resources and metric data.	Uninstalling the Monitor removes the application, as well as the resources and collected metrics. When upgrading, you need to install the new version of the Monitor, migrate your data, and then uninstall the old version. However, if you regularly backed up your Monitor, you can use the Migrate utility to migrate the data from the backed up files to the new version of the Monitor. See "Back up the Monitor" on page 240.
When monitoring a database that is part of database mirroring system, you receive errors about not being able to modify a read-only database.	In the **Administration** window, select the mirrored database resource and click **Edit**. In the **Other** field, type **NODE=PRIMARY**. See "NodeType (NODE) connection parameter" [*SQL Anywhere Server - Database Administration*].

The Relay Server

The Relay Server enables secure, load-balanced communication between mobile devices and back-end servers through a web server. Supported back-end servers include MobiLink, Unwired Server, Afaria, and Mobile Office. The Relay Server provides the following:

- A common communication architecture for mobile devices communicating with back-end servers.
- A mechanism to enable a load-balanced and fault-tolerant environment for back-end servers.
- A way to help communication between mobile devices and back-end servers in a way that integrates easily with existing corporate firewall configurations and policies.

For more information about the Relay Server, see "Relay Server".

MobiLink file-based download

File-based download is an alternative way to download data to SQL Anywhere remote databases: downloads can be distributed as files, enabling offline distribution of synchronization changes. This allows you to create a file once and distribute it to many remote databases.

With file-based download, you can put download synchronization changes in a file and transfer it to SQL Anywhere remote databases in any way a file can be transferred. For example, you can:

- broadcast the data by satellite multicast
- apply the update using Sybase Afaria
- email or FTP the file to users

You choose the users you want to receive the file. Full synchronization integrity is preserved in file-based download, including conflict detection and resolution. You can ensure that the file is secure by applying third-party encryption on the file.

When to use

File-based downloads are useful when a large amount of data changes on the consolidated database, but the remote database does not update the data frequently or does not do any updates at all. For example, price lists, product lists, and code tables.

File-based downloads are not useful when the downloaded data is updated frequently on the remote database or when you are running frequent upload-only synchronizations. In these situations, the remote sites may be unable to apply download files because of integrity checks that are performed when download files are applied.

File-based downloads currently can be used only with SQL Anywhere remote databases.

Download-only publications

Usually you should use a download-only publication for your file-based download. Use a regular publication only when you need to perform uploads with the same publication as you perform file-based downloads.

See “Download-only publications” [*MobiLink - Client Administration*].

If you use a regular publication, file-based downloads cannot be used as the sole means of updating remote databases. In that case you still need to regularly perform full synchronizations or upload-only synchronizations. Full or upload-only synchronizations are required to advance log offsets and to maintain the log file, which otherwise grows large and slows down synchronization. A full synchronization may also be required to recover from errors.

Setting up file-based download

The following steps provide an overview of the tasks required to set up file-based download, assuming that you already have MobiLink synchronization set up.

Usually you should use a download-only publication with your file-based download. Use a regular publication only when you need to do uploads with the same publication as you do file-based downloads.

Overview of setting up file-based download

1. Create a file-definition database.

   See "Creating the file-definition database" on page 248.

2. At the consolidated database, create scripts with a new script version.

   See "Changes at the consolidated database" on page 248.

3. Create a download file.

   See "Creating the download file" on page 249.

4. Apply the download file.

   See "Synchronizing new remotes" on page 250.

Other resources for getting started

- "File-based download examples" on page 254

Creating the file-definition database

To set up file-based download, you create a **file-definition database**. This is a SQL Anywhere database that has the same synchronization tables and publications as your remote databases. It can be located anywhere. This database contains no data or state information. It does not have to be backed up or maintained; in fact, you can delete it and recreate it as needed.

The file-definition database must include the following:

- the same publications as the remote databases, the tables and columns used in the publication, the foreign key relationships and constraints of those tables and columns, and the tables required by those foreign key relationships.
- a MobiLink user name that identifies the group of remote databases that are to apply the download file. You use this group MobiLink user name in your synchronization scripts to identify the group of remote databases.

Changes at the consolidated database

On the consolidated database, create a new script version for your file-based download and implement any scripts required by your existing synchronization system into it. Upload scripts are not required. This

script version is used only for file-based download. For this script version, all scripts that take MobiLink user names as parameters, instead, take a MobiLink user name that refers to a group of remote databases. This is the user name that is defined in the file-definition database.

For each script version that you have defined, implement a begin_publication script.

For timestamp-based downloads, implement a modify_last_download_timestamp script for each script version. How you implement this script depends on how much data you intend to send in each download file. For example, one approach is to use the earliest time that any user from the group last downloaded successfully. Remember that the ml_username parameter passed to this script is actually the group name.

Note
It is strongly recommend that you use the -dsd option on the MobiLink server when generating file-based download files from a Microsoft SQL Server consolidated database. If you do not use the -dsd option, remotes may occasionally be unable to apply a file-based download file and will report an error similar to the following: `The last download time for publication <publication> is <timestamp> The download file's next last download time was <timestamp> Cannot apply a download file if its next last download time is before the publication's last download time.`

Generally, the frequency with which a remote is unable to apply a download file will be proportional to the frequency with which the remote performs normal (connected) synchronizations and the amount of concurrent activity in the consolidated database.

See "-dsd mlsrv12 option" on page 46.

See also

- "Script versions" on page 282
- "begin_publication connection event" on page 332
- "modify_last_download_timestamp connection event" on page 402

Creating the download file

The download file contains the data to be synchronized. To create the download file, set up your file-definition database and consolidated database as described above. Run dbmlsync with the -bc option and supply a file name with the extension *.df*. For example,

```
dbmlsync -c "uid=DBA;pwd=sql;server=fbdl_eng;dbf=fdef.db" -v+
   -e "sv=filebased" -bc file1.df
```

You can also choose to specify options when you create the download file:

- **-be option** Use -be to add a string to the download file that can be accessed at the remote database using the sp_hook_dbmlsync_validate_download_file stored procedure.

  See "-be dbmlsync option" [*MobiLink - Client Administration*] and "sp_hook_dbmlsync_validate_download_file" [*MobiLink - Client Administration*].

- **-bg option** Use the -bg option to create a download file that can be used by remotes that have never synchronized.

Synchronizing new remotes

If you want to apply a download file to a remote database that has never synchronized using MobiLink, then before you apply the download file you need to either perform a normal synchronization on the remote database or use the dbmlsync -bg option when creating the download file.

For timestamp-based synchronization, doing either of these two things causes the download of an initial snapshot of the data. For both timestamp and snapshot based synchronization, this step sets the generation number to the value that is generated by the begin_publication script on the consolidated database.

Perform a normal synchronization

You can prepare a remote database to receive download files by performing a synchronization that does not use a download file.

Use the -bg option

Alternatively, you can create a download file with the -bg option to use with remotes that haven't yet synchronized. You apply this initial download file to prepare the remote database for file-based synchronization.

- **Snapshot downloads** If you are performing snapshot downloads, then the initial download file just needs to set the generation number. You may choose to include an initial snapshot of the data in this file, but since each snapshot download contains all the data and does not depend on previous downloads, this is not required.

  For snapshot downloads, using the -bg option is straightforward. Just specify -bg in the dbmlsync command line when you create the download file. You can use the same script version to create the initial download file as you use for subsequent download files.

- **Timestamp-based downloads** If you are performing timestamp-based downloads, then the initial download must set the generation number on the remote database and include a snapshot of the data. With timestamp-based downloads, each download builds on previous ones. Each download file contains a last download timestamp. All rows changed on the consolidated after the file's last download timestamp are included in the file. To apply a file, a remote database must already have received all the changes that occurred before the file's last download timestamp. This is confirmed by checking that the file's last download timestamp is greater than or equal to the remote database's last download timestamp (the time up to which the remote database has received all changes from the consolidated database).

  Before a remote can apply its first normal download file, it must receive all data changed before that file's last download timestamp and after January 1, 1900. The initial download file created with the -bg option must contain this data. The easiest way to select this data is to create a separate script version that uses the same download_cursor's as your normal file-based synchronization script version

but does not have a modify_last_download_timestamp script. If no modify_last_download_timestamp script is defined, then the last download timestamp for a file-based download defaults to January 1, 1900.

If you apply download files built with the -bg option to remote databases that have already synchronized, the -bg option causes the generation numbers on the remote database to be updated with the value on the consolidated database at the time the download file was created. This defeats the purpose of generation numbers, which is to prevent you from applying further file-based downloads until an upload has been performed in situations such as when recovering a consolidated database that is lost or corrupted.

See "MobiLink generation numbers" on page 253.

Validation checks

Before applying a download file to a remote database, dbmlsync does several things to ensure that the synchronization is valid.

- dbmlsync checks the download file to ensure that the file-definition database that was used to create it has:
  - the same publication as the remote database
  - the same tables and columns used in the publication
  - the same foreign key relationships and constraints as those tables and columns
- dbmlsync checks to see if there is any data in the publication that has not been uploaded from the remote. If there is, the download file is not applied, because applying the download file could cause pending upload data to be lost.
- dbmlsync checks the last download timestamp, next last download timestamp, and creation time of the download file to ensure that:
  - newer data on the remote database is not overwritten by older data contained in the download file.
  - a download file is not applied if applying it means that the remote database would miss some changes that have occurred on the consolidated database. This situation might occur if the remote did not apply previous file-based downloads.

    See "Automatic validation" on page 252.
- Optionally, dbmlsync checks the generation number in the remote database to ensure it matches the generation number in the download file.

  See "MobiLink generation numbers" on page 253.
- Optionally, you can create custom validation logic with the sp_hook_dbmlsync_validate_download_file stored procedure.

  For more information, see "Custom validation" on page 253.

Automatic validation

Before applying a download file, dbmlsync performs special checks on the last download timestamp, next last download timestamp, download file creation time, and transaction log.

Last download timestamp and next last download timestamp

Each download file contains all changes to be downloaded that occurred on the consolidated database between the file's last download timestamp, and its next last download timestamp. The time at the consolidated database is used for both time values. By default the file's last download time is Jan 1, 1900 12:00 AM and the file's next last download timestamp is the time the download file was created. These defaults can be overridden by implementing the generate_next_last_download_timestamp, modify_last_download_timestamp, and modify_next_last_download_timestamp scripts on the consolidated database.

A remote site can apply a download file only if the file's last download timestamp is less than or equal to the remote's last download timestamp. This ensures that a remote never misses operations that occur on the consolidated database. Usually when a file-based download fails based on this check, the remote has missed one or more download files. The situation can be corrected by applying the missing download files or by performing a full or download-only synchronization.

In addition, a remote site can apply a download file only if the file's next last download timestamp is greater than the remote's last download timestamp. The remote's last download timestamp is the time (at the consolidated database) up to which the remote has received all changes that are to be downloaded. The remote database's last download time is updated each time the remote successfully applies a download (normal or file-based). This check ensures that a download file is not applied if more recent data has already been downloaded. A common case where this could happen occurs when download files are applied out of order. For example, suppose a download file *F1.df* is created, and another file *F2.df* is created later. This check ensures that *F1.df* cannot be applied after *F2.df*, because that could allow newer data in *F2.df* to be overwritten with older data in *F1.df*.

When a file-based download fails based on the next last download timestamp, no additional action is required other than to delete the file. Synchronization succeeds once a new file is received.

Creation time

The download file's creation time indicates the time at the consolidated database when creation of the file began. A download file can only be applied if the file's creation time is greater than the remote database's last upload time. The remote's last upload time is the time at the consolidated database when the remote's last successful upload was committed. This check ensures that data that has been uploaded after the creation of the download (and is newer than the download) is not overwritten by older data in the download file.

When a download file is rejected based on this check, no action is required. The remote site should be able to apply the next download file.

When an upload fails because dbmlsync did not receive an acknowledgement after sending an upload to the MobiLink server, the remote database's last upload time may be incorrect. In this case, the creation time check cannot be performed and the remote is unable to apply download files until it completes a normal synchronization.

Transaction log

Before applying a download file, dbmlsync scans the remote database's transaction log and builds up a list of all changes that must be uploaded. Dbmlsync only applies a download file if it does not contain any operations that affect rows with changes that must be uploaded.

MobiLink generation numbers

Generation numbers provide a mechanism for forcing remote databases to upload data before applying any more download files. This is especially useful when a problem on the consolidated database has resulted in data loss and you must recover lost data from the remote databases.

On the remote database, a separate generation number is automatically maintained for each subscription. On the consolidated database, the generation number for each subscription is determined by the begin_publication script. Each time a remote performs a successful upload, it updates the remote generation number with the value set by the begin_publication script in the consolidated database.

Each time a download file is created, the generation number set by the begin_publication script is stored in the download file. A remote site only applies a download file if the generation number in the file is equal to the generation number stored in the remote database.

Note
Whenever the generation number generated by the begin_publication script for a file-based download changes, the remote databases must perform a successful upload before they can apply any new download files.

The sp_hook_dbmlsync_validate_download_file stored procedure can be used to override the default checking of the generation number.

For more information about managing MobiLink generation numbers, see:

- “begin_publication connection event” on page 332
- “end_publication connection event” on page 365
- “sp_hook_dbmlsync_validate_download_file” [*MobiLink - Client Administration*]

Custom validation

You can create custom validation logic to determine if a download file should be applied to a remote database. You do this with the sp_hook_dbmlsync_validate_download_file stored procedure. With this stored procedure, you can both reject a download file and override the default checking of the generation number.

You can use the dbmlsync -be option to embed a string in the file. You use the -be option against the file-definition database when you create the download file This string is passed to the sp_hook_dbmlsync_validate_download_file through the #hook_dict table, and can be used in your validation logic.

For more information, see "sp_hook_dbmlsync_validate_download_file" [*MobiLink - Client Administration*].

File-based download examples

This section contains two examples. Each sets up a file-based download synchronization using a consolidated database with only one table. The first is a simple snapshot example and the second is a slightly more involved timestamp-based example.

Snapshot example

This example implements file-based download for snapshot synchronization. It sets up the three databases that are required by the file-based download, and then demonstrates how to download data. This example is presented in such a way that you can either just read through it, or you can cut and paste the text to run the sample.

Create databases for the sample

The following commands create the three databases used in the example: a consolidated database, a remote database, and a file-definition database.

```
dbinit scons.db
dbinit sremote.db
dbinit sfdef.db
```

The following commands start the three databases and create a data source name for MobiLink to use to connect to the consolidated database.

```
dbeng12 -n sfdef_eng sfdef.db
dbeng12 -n scons_eng scons.db
dbeng12 -n sremote_eng stremote.db
dbdsn -y -w fbd_demo -c "server=scons_eng;dbf=scons.db;uid=DBA;
   pwd=sql;astart=off;astop=off"
```

Open Interactive SQL, connect to *scons.db* and run the MobiLink setup script. For example:

```
read "C:\Program Files\SQL Anywhere 12\MobiLink\setup\syncsa.sql"
```

Start the MobiLink server:

```
start mlsrv12 -v+ -c "dsn=fbd_demo" -zu+ -ot scons.txt
```

Set up the snapshot example consolidated database

In this example, the consolidated database has one table, called T1. After connecting to the consolidated database, you can run the following SQL to create table T1:

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);
```


The following code creates a script version called filebased and creates a download script for that script version.

```
CALL ml_add_table_script( 'filebased',
  'T1', 'download_cursor',
  'SELECT pk, c1 FROM T1' );
```

The following code creates a script version called normal and creates upload and download scripts for that script version.

```
CALL ml_add_table_script ( 'normal', 'T1',
  'upload_insert',
  'INSERT INTO T1 VALUES ({ml r.pk}, {ml r.c1})');

CALL ml_add_table_script( 'normal', 'T1',
  'upload_update',
  'UPDATE T1 SET c1 = {ml r.c1} WHERE pk = {ml r.pk} ' );

CALL ml_add_table_script( 'normal', 'T1',
 'upload_delete',
 'DELETE FROM T1 WHERE pk = {ml r.pk}' );

CALL ml_add_table_script( 'normal', 'T1',
  'download_cursor',
  'SELECT pk, c1 FROM T1' );

COMMIT;
```

The following command creates the stored procedure begin_pub and specifies that begin_pub is the begin_publication script for both the "normal" and "filebased" script versions:

```
CREATE PROCEDURE begin_pub (
       INOUT generation_num integer,
       IN    username       varchar(128),
       IN    pubname        varchar(128) )
BEGIN
   SET generation_num=1;
END;

CALL ml_add_connection_script(
  'filebased',
  'begin_publication',
  '{ call begin_pub(
      {ml s.generation_number},
      {ml s.username},
      {ml s.publication_name} ) }' );

CALL ml_add_connection_script( 'normal',
  'begin_publication',
  '{ call begin_pub(
      {ml s.generation_number},
      {ml s.username},
      {ml s.publication_name} ) }' );
```

Create the snapshot example remote database

In this example, the remote database also contains one table, called T1. Connect to the remote database and run the following SQL to create the table T1, a publication called P1, and a user called U1. The SQL also creates a subscription for U1 to P1.

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
```

```
 c1 INTEGER
);

CREATE PUBLICATION P1 (
    TABLE T1
);

CREATE SYNCHRONIZATION USER U1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR U1;
```

The following code creates an sp_hook_dbmlsync_validate_download_file hook to implement user-defined validation logic in the remote database:

```
CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file()
BEGIN
    DECLARE udata varchar(256);
SELECT value
    INTO udata
    FROM #hook_dict
    WHERE name = 'user data';
    IF udata <> 'ok' THEN
    UPDATE #hook_dict
    SET value = 'FALSE'
    WHERE name = 'apply file';
    END IF;
END
```

Create the snapshot example file-definition database

A file-definition database is required in MobiLink systems that use file-based download. This database has the same schema as the remote databases being updated by file-based download, and it contains no data or state information. The file-definition database is used solely to define the structure of the data that is to be included in the download file. One file-definition database can be used for many groups of remote databases, each defined by its own MobiLink group user name.

The following code defines the file-definition database for this sample. It creates a schema that is identical to the remote database, and also creates:

- a publication called P1 that publishes all rows of the T1 table. The same publication name must be used in the file-definition database and the remote databases.
- a MobiLink user called G1. This user represents all the remotes that are to be updated in the file-based download.
- a subscription to the publication.

You must connect to sfdef.db before running this code.

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);

CREATE PUBLICATION P1 (
    TABLE T1
```



```
);

CREATE SYNCHRONIZATION USER G1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR G1;
```

Prepare for initial synchronization

To prepare your new remote database so that you can apply a download file, you need to either perform a normal synchronization or create the download file with the dbmlsync -bg option. This example shows you how to initialize your new remote database by performing a normal synchronization.

You can perform an initial synchronization of the remote database with the script version called normal that was created earlier:

```
dbmlsync -c "uid=DBA;pwd=sql;server=sremote_eng;
  dbf=sremote.db" -v+ -e "sv=normal"
```

Demonstrate the snapshot example file-based download

Connect to the consolidated database and insert some data that is synchronized by file-based download, such as the following:

```
INSERT INTO T1 VALUES( 1, 1 );
INSERT INTO T1 VALUES( 2, 4 );
INSERT INTO T1 VALUES( 3, 9 );
COMMIT;
```

The following command must be run on the computer that holds the file-definition database. It does the following:

- The dbmlsync -bc option creates the download file, and names it *file1.df*.
- The -be option includes the string "OK" in the download file that is accessible to the sp_dbmlsync_validate_download_file hook.

```
dbmlsync -c
   "uid=DBA;pwd=sql;server=sfdef_eng;dbf=sfdef.db"
   -v+ -e "sv=filebased" -bc file1.df -be ok -ot fdef.txt
```

To apply the download file, run dbmlsync with the -ba option on the remote database, supplying the name of the download file you want to apply:

```
dbmlsync -c "uid=DBA;pwd=sql;server=sremote_eng;
   dbf=sremote.db" -v+ -ba file1.df -ot remote.txt
```

The changes are now applied to the remote database. Open Interactive SQL, connect to the remote database, and run the following SQL statement to verify that the remote has the data:

```
SELECT * FROM T1
```

Clean up the snapshot example

The following commands stop all three database servers and erase the files.

```
del file1.df
mlstop -h -w
```

```
dbstop -y -c "server=sfdef_eng; uid=DBA; pwd=sql"
dbstop -y -c "server=scons_eng; uid=DBA; pwd=sql"
dbstop -y -c "server=sremote_eng; uid=DBA; pwd=sql"
dberase -y sfdef.db
dberase -y scons.db
dberase -y sremote.db
```

Timestamp-based example

This example implements file-based download for timestamp-based synchronization. It sets up the three databases and then demonstrates how to download data by file. This example is presented in such a way that you can either just read through it, or you can cut and paste the text to run the sample.

Create databases for the sample

The following commands create the three databases used in the example: a consolidated database, a remote database, and a file-definition database.

```
dbinit tcons.db
dbinit tremote.db
dbinit tfdef.db
```

The following commands start the three databases and create a data source name for MobiLink to use to connect to the consolidated database.

```
dbeng12 -n tfdef_eng tfdef.db
dbeng12 -n tcons_eng tcons.db
dbeng12 -n tremote_eng tremote.db
dbdsn -y -w tfbd_demo -c "server=tcons_eng;dbf=tcons.db;uid=DBA;
   pwd=sql;astart=off;astop=off"
```

Open Interactive SQL, connect to *tcons.db* and run the MobiLink setup script. For example:

```
read "C:\Program Files\SQL Anywhere 12\MobiLink\setup\syncsa.sql"
```

Start the MobiLink server:

```
start mlsrv12 -v+ -c "dsn=tfbd_demo" -zu+ -ot tcons.txt
```

Set up the timestamp example consolidated database

In this example, the consolidated database has one table, called T1. After connecting to the consolidated database, you can run the following code to create T1:

```
CREATE TABLE T1 (
 pk  INTEGER PRIMARY KEY,
 c1  INTEGER,
 last_modified TIMESTAMP DEFAULT TIMESTAMP
);
```

The following code defines a script version called normal with a minimal number of scripts. This script version is used for synchronizations that do **not** use file-based download.

```
CALL ml_add_table_script( 'normal', 'T1',
   'upload_insert',
   'INSERT INTO T1( pk, c1) VALUES( {ml r.pk}, {ml r.c1} )' );
```



```
CALL ml_add_table_script( 'normal', 'T1',
   'upload_update',
   'UPDATE T1 SET c1 = {ml r.c1} WHERE pk = {ml r.pk} ' );

CALL ml_add_table_script( 'normal', 'T1',
   'upload_delete',
   'DELETE FROM T1 WHERE pk = {ml r.pk}' );

CALL ml_add_table_script( 'normal', 'T1',
   'download_cursor',
   'SELECT pk, c1 FROM T1
      WHERE last_modified >= {ml s.last_table_download}' );
```

The following code sets the generation number for all subscriptions to 1. Generation numbers can be useful if your consolidated database becomes lost or corrupted and you need to force an upload.

```
CREATE PROCEDURE begin_pub (
       INOUT generation_num integer,
       IN    username       varchar(128),
       IN    pubname        varchar(128) )
BEGIN
 SET generation_num = 1;
END;

CALL ml_add_connection_script( 'normal',
    'begin_publication',
    '{ call begin_pub(
        {ml s.generation_number},
        {ml s.username},
        {ml s.publication_name},
        {ml s.last_publication_upload},
        {ml s.last_publication_download} ) }' );

COMMIT;
```

The following code defines the script version called filebased. This script version is used to create file-based download.

```
CALL ml_add_connection_script( 'filebased',
    'begin_publication',
    '{ call begin_pub(
        {ml s.generation_number},
        {ml s.username},
        {ml s.publication_name} ) }' );

CALL ml_add_table_script( 'filebased', 'T1',
    'download_cursor',
    'SELECT pk, c1 FROM T1
        WHERE last_modified >= {ml s.last_table_download}' );
```

The following code sets the last download time so that all changes that occurred within the last five days are included in download files. Any remote that has missed all the download files created in the last five days have to perform a normal synchronization before being able to apply any more file-based downloads.

```
CREATE PROCEDURE ModifyLastDownloadTimestamp(
       INOUT last_download_timestamp TIMESTAMP,
       IN    ml_username             VARCHAR(128) )
BEGIN
    SELECT dateadd( day, -5, CURRENT TIMESTAMP )
    INTO last_download_timestamp;
END;
```

```
CALL ml_add_connection_script( 'filebased',
   'modify_last_download_timestamp',
   'CALL ModifyLastDownloadTimestamp(
      {ml s.last_download}, {ml s.username} )' );

COMMIT;
```

Create the timestamp example remote database

In this example, the remote database also contains one table, called T1. After connecting to the remote database, run the following code to create table T1, a publication called P1, and a user called U1. The code also creates a subscription for U1 to P1.

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);

CREATE PUBLICATION P1 (
    TABLE T1
);

CREATE SYNCHRONIZATION USER U1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR U1;
```

The following code defines an sp_hook_dbmlsync_validate_download_file stored procedure. This stored procedure prevents the application of download files that do not have the string "ok" embedded in them.

```
CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file()
BEGIN
    DECLARE udata varchar(256);

    SELECT value
    INTO udata
    FROM #hook_dict
    WHERE name = 'user data';

    IF udata <> 'ok' THEN
      UPDATE #hook_dict
      SET value = 'FALSE'
      WHERE name = 'apply file';
    END IF;
END
```

Create the timestamp example file-definition database

The following code defines the file-definition database for the timestamp example. It creates a table, a publication, a user, and a subscription for the user to the publication.

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);

CREATE PUBLICATION P1 (
    TABLE T1
);
```



```
CREATE SYNCHRONIZATION USER G1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR G1;
```

Prepare for initial synchronization

To prepare your new remote database so that you can apply a download file, you need to either perform a normal synchronization or create the download file with the dbmlsync -bg option. This example shows you how to use -bg.

The following code defines a script version called filebased_init for the consolidated database. This script version has a single begin_publication script.

```
CALL ml_add_table_script(
  'filebased_init', 'T1', 'download_cursor',
  'SELECT pk, c1 FROM T1' );

CALL ml_add_connection_script(
  'filebased_init',
  'begin_publication',
  '{ call begin_pub(
      {ml s.generation_number},
      {ml s.username},
      {ml s.publication_name} ) }' );

COMMIT;
```

The following two command lines create and apply an initial download file using the script version called filebased_init and the -bg option.

```
dbmlsync -c "uid=DBA;pwd=sql;server=tfdef_eng;dbf=tfdef.db"
  -v+ -e "sv=filebased_init" -bc tfile1.df -be ok -bg
  -ot tfdef1.txt

dbmlsync -c "uid=DBA;pwd=sql;server=tremote_eng;dbf=tremote.db"
  -v+ -ba tfile1.df -ot tremote.txt
```

Demonstrate the timestamp example file-based download

Connect to the consolidated database and insert some data that is synchronized by file-based download, such as the following:

```
INSERT INTO T1(pk, c1) VALUES( 1, 1 );
INSERT INTO T1(pk, c1) VALUES( 2, 4 );
INSERT INTO T1(pk, c1) VALUES( 3, 9 );
commit;
```

The following command line creates a download file containing the new data.

```
dbmlsync -c
   "uid=DBA;pwd=sql;server=tfdef_eng;dbf=tfdef.db"
   -v+ -e "sv=filebased" -bc tfile2.df -be ok -ot tfdef2.txt
```

The following command line applies the download file to the remote database.

```
dbmlsync -c "uid=DBA;pwd=sql;server=tremote_eng;dbf=tremote.db"
   -v+ -ba tfile2.df -ot tfdef3.txt
```

The changes are now applied to the remote database. Open Interactive SQL, connect to the remote database, and run the following SQL statement to verify that the remote has the data:

```
SELECT * FROM T1
```

Clean up the timestamp example

The following commands stop all three database servers and then erase the files.

```
del tfile1.df
mlstop -h -w
dbstop -y -c "server=tfdef_eng; uid=DBA; pwd=sql"
dbstop -y -c "server=tcons_eng; uid=DBA; pwd=sql"
dbstop -y -c "server=tremote_eng; uid=DBA; pwd=sql"
dberase -y tfdef.db
dberase -y tcons.db
dberase -y tremote.db
```


MobiLink events

This section describes how to write scripts for MobiLink events.

Writing synchronization scripts

You control the synchronization process by writing synchronization scripts and storing or referencing them in MobiLink system tables in the consolidated database. You can write scripts in SQL, Java, or .NET.

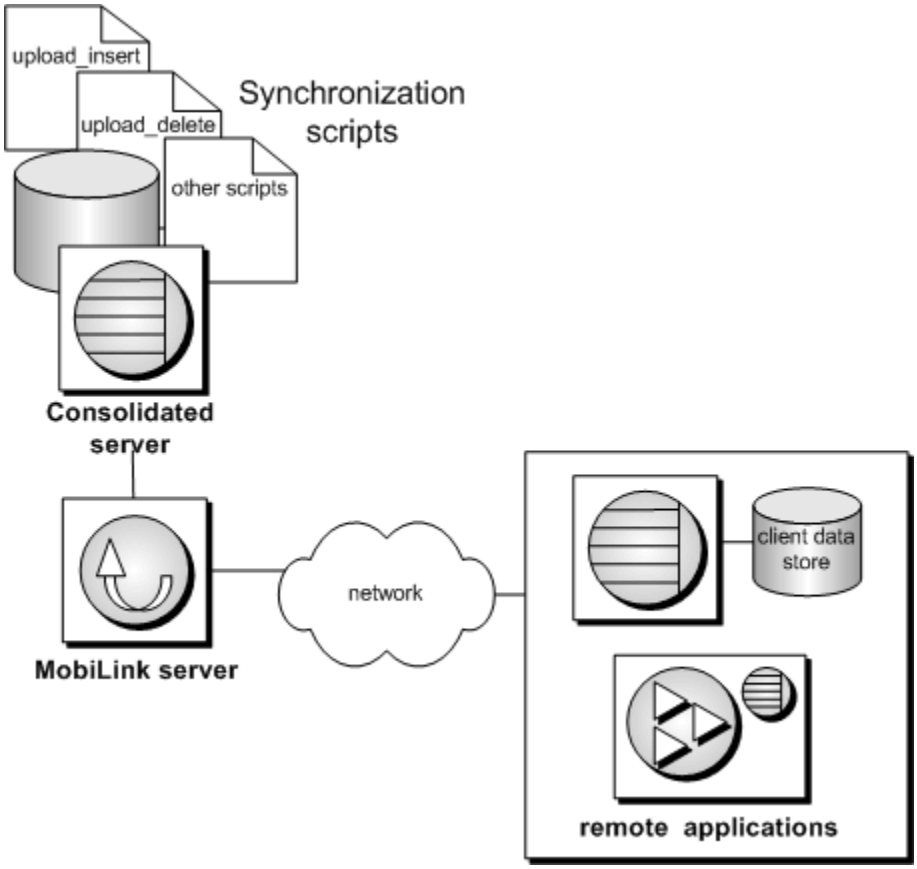
MobiLink synchronization logic is specified with synchronization scripts. Scripts define:

- how data that is uploaded from the remote database should be applied to the consolidated database
- what data should be downloaded from the consolidated database
- how authentication takes place during synchronization (optional)

Scripts can be individual statements or stored procedure calls. They are stored or referenced in your consolidated database. To add scripts to the consolidated database, you can use Sybase Central or you can use system procedures.

Caution
There should be no implicit or explicit commit or rollback in your SQL synchronization scripts or the procedures or triggers that are called from your SQL synchronization scripts. COMMIT or ROLLBACK statements within SQL scripts alter the transactional nature of the synchronization steps. If you use them, MobiLink cannot guarantee the integrity of your data in the event of a failure.

During synchronization, the MobiLink server reads the scripts if they are not already loaded, then executes them against the consolidated database.



The synchronization process has multiple steps. A unique event identifies each step. You control the synchronization process by writing scripts associated with some of these events. You write a script only when some particular action must occur at a particular event. The MobiLink server executes each script when its associated event occurs. If you do not define a script for a particular event, the MobiLink server simply proceeds to the next step.

For example, one event is begin_upload_rows. You can write a script and associate it with this event. The MobiLink server reads this script when it is first needed, and executes it during the upload phase of synchronization. If you write no script, the MobiLink server proceeds immediately to the next step, which is processing the uploaded rows.

Some scripts, called table scripts, are associated not only with an event, but also with a particular table in the remote database. The MobiLink server performs some tasks on a table-by-table basis; for example, downloading rows. You can have many scripts associated with the same event, but each with different application tables. Alternatively, you can define many scripts for some application tables, and very few for others.

For an overview of events, see "The synchronization process" [*MobiLink - Getting Started*].

For a description of every script you can write, see "Synchronization events" on page 297.

You can write scripts in SQL, Java, or .NET. This section applies to all kinds of scripts, but focuses on how to write synchronization scripts in SQL.

For a description and comparison of SQL, Java, and .NET, see "Options for writing server-side synchronization logic" [*MobiLink - Getting Started*].

For information about writing scripts in .NET, see "Writing synchronization scripts in .NET" on page 588.

For information about writing scripts in Java, see "Writing synchronization scripts in Java" on page 523.

For information about how to implement synchronization scripts, see "Synchronization techniques" on page 86.

Simple synchronization script

MobiLink provides many events that you can exploit, but it is not mandatory to provide scripts for each event. In a simple synchronization model, you may need only a few scripts.

Downloading all the rows from the table to each remote database synchronizes the ULProduct table in the CustDB sample application. In this case, no additions are permitted at the remote databases. You can implement this simple form of synchronization with two scripts; in this case only two events have a script associated with them.

The MobiLink event that controls the rows to be downloaded during each synchronization is named the download_cursor event. Cursor scripts must contain SELECT statements. The MobiLink server uses these queries to define a cursor. For a download_cursor script, the cursor selects the rows to download to one particular table in the remote database.

In the CustDB sample application, there is a single download_cursor script for the ULProduct table in the sample application, which consists of the following query:

```
SELECT prod_id, price, prod_name
FROM ULProduct
```

This query generates a result set. The rows that make up this result set are downloaded to the client. In this case, all the rows of the table are downloaded.

The MobiLink server knows to send the rows to the ULProduct application table because this script is associated with both the download_cursor event and ULProduct table by the way it is stored in the consolidated database. Sybase Central allows you to make these associations.

The second required event is the download_delete_cursor, which must have a script defined, along with the download_cursor, for each table being downloaded. In this simple example, we do not perform download deletes so we simply define the script as --{ml_ignore}.

In this example, the query selects data from a consolidated table also named ULProduct. The names need not match. You could, instead, download data to the ULProduct application table from any table, or any combination of tables, in the consolidated database by rewriting the query.

You can write more complicated synchronization scripts. For example, you could write a script that downloads only recently modified rows, or one that provides different information to each remote database.

Scripts and the synchronization process

Each script corresponds to a particular event in the synchronization process. You write a script only when some action must occur. All unnecessary events can be left undefined.

The two principal parts of the process are the processing of uploaded information and the preparation of rows for downloading. If rows are uploaded from a remote table you must define the appropriate upload script(s). If a table is to have rows downloaded via SQL then both the download_cursor and download_delete_cursor scripts must be defined.

The MobiLink server reads and prepares each script once, when it is first needed. The script is then executed whenever the event is invoked.

The sequence of events

For information about the full sequence of MobiLink events, see "Overview of MobiLink events" on page 297.

For the details of upload processing, see "Writing scripts to upload rows" on page 288.

For the details of download processing, see "Writing scripts to download rows" on page 290.

Notes

- MobiLink technology allows multiple clients to synchronize concurrently. In this case, each client uses a separate connection to the consolidated database.
- The begin_connection and end_connection events are independent of any one synchronization as one connection can handle many synchronization requests. These scripts have no parameters. These are examples of connection-level scripts.
- Some events are invoked only once for each synchronization regardless of how many tables are synchronized. These are connection-level scripts.
- Some events are invoked once for each table being synchronized. Scripts associated with these events are called table-level scripts.

  While each table can have its own table scripts, you can also write table-level scripts that are shared by several tables, though this is uncommon.
- Some events, such as begin_synchronization, occur at both the connection level and the table level. You can supply both connection and table scripts for these events.

For reference material, including details about each script and its parameters, see "Synchronization events" on page 297.

Script types

There are two types of synchronization scripts:

- **connection-level scripts** These scripts perform actions that are connection-specific or synchronization-specific and that are independent of any one remote table. These scripts can be used in conjunction with other scripts to implement your synchronization business logic.

  See "Connection scripts" on page 267.

- **table-level scripts** These scripts perform actions specific to one synchronization and one particular remote table. These scripts are used in conjunction with other scripts to implement your synchronization business logic, including conflict resolution.

  See "Table scripts" on page 267.

Connection scripts

Connection-level 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.

Connection scripts control actions centered on connecting and disconnecting, and synchronization-level event actions such as beginning and ending the upload or download process. Some connection scripts have related table scripts. These connection scripts are always invoked regardless of the tables being synchronized.

You only need to write a connection-level script when some action must occur at a particular event. You may need to create scripts for only a few events. The default action at any event is for the MobiLink server to perform no actions. Some simple synchronization schemes need no connection scripts.

ml_global script version

To save you from defining the same scripts multiple times, you can define connection-level scripts once and then re-use them from any script version. You do this by defining a script version called ml_global.

See "ml_global script version" on page 283.

Table scripts

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

The synchronization scripts for a given table can refer to any table (or a combination of tables) in the consolidated database. You can use this feature to fill a particular remote table with data stored in one or more consolidated tables, or to store data uploaded from a single remote table into multiple tables in the consolidated database.

Table names need not match

The names of tables in the remote databases need not match the names of the tables in the consolidated database. The MobiLink server determines which scripts are associated with a table by looking up the remote table name in the ml_table system table. The scripts themselves reference the consolidated tables of your choice.

Script parameters

Most synchronization scripts can receive parameters from the MobiLink server. For details about the parameters you can use in each script, see "Synchronization events" on page 297.

You can specify parameters in your SQL scripts in one of two ways:

- named script parameters
- question marks (deprecated in SQL scripts)

Named script parameters

Named parameters have the following advantages over (deprecated) question marks:

- Named parameters allow you to specify any subset of the available parameters in any order.
- With the exception of in/out parameters, you can specify the same named parameter more than once within a script.
- When you use named parameters, you can specify the remote ID in your scripts. This is the only way to specify the remote ID in scripts.
- You can create your own named parameters. See "User-defined named parameters" on page 280.

You cannot mix named parameters and question marks in a single script.

There are four types of MobiLink named parameters. To specify a named parameter, you must prefix it with its type, as follows:

Type of named parameter	Prefix	Examples
System parameters.	s.	{ml s.remote_id}
Row parameters. (The column name. If the column contains spaces, enclose it in double quotes or square brackets.)	r.	{ml r.cust_id} {ml r."Column name"}

Type of named parameter	Prefix	Examples
Old row parameters. (Only used in upload_update scripts to specify the pre-image column values. If the column name contains spaces, enclose it in double quotes or square brackets.)	o.	{ml o.cust_name} {ml o."Column name"}
Authentication parameters. See "Authentication parameters" on page 281.	a.	{ml a.1}
User-defined parameters. See "User-defined named parameters" on page 280.	u. or ui.	• **{ml u.varname}** Use if the parameter if the parameter will be updated. User-defined parameters prefixed with u. are in/out. • **{ml ui.varname}** Use if the parameter will only be referenced. User-defined parameters prefixed with ui. are input-only.

To reference a script parameter by name, enclose the parameter in curly braces and prefix it with ml, as in **{ml** *parameter* **}**. For example, **{ml s.action_code}**. The curly brace notation is an ODBC convention.

For convenience, you can enclose a larger section of code in the curly braces, as long as the section of code does not contain any schema names with the same name as a MobiLink script parameter. For example, each of the following upload_insert scripts are valid and equivalent:

```
INSERT INTO t ( id, c0 ) VALUES( {ml r.id}, {ml r.c0} )
```

and

```
INSERT INTO t ( is, c0 ) VALUES({ml r.id, r.c0})
```

and

```
{ml  INSERT INTO t ( id, c0 ) VALUES( r.id, r.c0 ) }
```

Note
To use named row parameters you may need to use the ml_add_column system procedure to store column information in the consolidated database, although most deployments using version 12 synchronization clients do not require this. See "ml_add_column system procedure" on page 693.

Script parameters represented by question marks (deprecated for SQL)

Note
Representing parameters with question marks has been deprecated in SQL scripts. It is strongly recommended that you use named parameters instead. See "Named script parameters" on page 268 and "User-defined named parameters" on page 280.

Representing parameters with question marks is an ODBC convention. To use question marks in your MobiLink SQL scripts, place a single question mark in your script for each parameter. The MobiLink server replaces each question mark with the value of a parameter. It substitutes values in the order the parameters appear in the script definition.

The parameters must be in the order specified in "Synchronization events" on page 297. Some parameters are optional. A parameter is optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

Commenting script parameters

The following forms of comments are recognized:

- Double hyphen prefix (--)
- Double forward slash prefix (//)
- Block commenting (/* */)

The first two forms cause the script text to be ignored until the end of a line. The third form causes all script text between the /* prefix and the */suffix to be ignored. Block commenting cannot be nested.

Any other type of vendor-specific comment is not recognized and should not be used to comment references to a named parameter.

MobiLink system parameters and events

System parameter	Event type	Event parameter is available in
action_code	connection	The action_code parameter is available in the following events: • handle_error • handle_odbc_error • report_error • report_odbc_error
authentication_status	connection	The authentication_status parameter is available in the following events: • authenticate_parameters • authenticate_user • authenticate_user_hashed

System parameter	Event type	Event parameter is available in
bytes	connection and table	The bytes parameter is available in the following events: • download_statistics • upload_statistics
conflicted_deletes	connection and table	The conflicted_deletes parameter is available in the following events: • upload_statistics
conflicted_inserts	connection and table	The conflicted_inserts parameter is available in the following events: • upload_statistics
conflicted_updates	connection and table	The conflicted_updates parameter is available in the following events: • upload_statistics
connection_retries	connection	The conflicted_retries parameter is available in the following events: • synchronization_statistics
deadlocks	connection and table	The deadlocks parameter is available in the following events: • synchronization_statistics (connection event) • upload_statistics (connection and table events)
deleted_rows	connection and table	The deleted_rows parameter is available in the following events: • download_statistics • upload_statistics
error_code	connection	The error_code parameter is available in the following events: • handle_error • modify_error_message • report_error

System parameter	Event type	Event parameter is available in
error_message	connection	The error_message parameter is available in the following events: ● handle_error ● handle_odbc_error ● modify_error_message ● report_error ● report_odbc_error
errors	connection and table	The errors parameter is available in the following events: ● download_statistics ● synchronization_statistics ● upload_statistics
event_name	connection and table	The event_name parameter is available in the following events: ● time_statistics
fetched_rows	connection and table	The fetched_rows parameter is available in the following events: ● download_statistics
file_authentication_code	connection	The file_authentication_code parameter is available in the following events: ● authenticate_file_transfer ● authenticate_file_upload
filename	connection	The filename parameter is available in the following events: ● authenticate_file_transfer ● authenticate_file_upload
file_size	connection	The file_size parameter is available in the following events: ● authenticate_file_upload
filtered_rows	connection and table	The filtered_rows parameter is available in the following events: ● download_statistics

System parameter	Event type	Event parameter is available in
generation_number	connection	The filtered_rows parameter is available in the following events: • begin_publication • end_publication
hashed_new_password	connection	The hashed_new_password parameter is available in the following events: • authenticate_user_hashed
hashed_password	connection	The hashed_password parameter is available in the following events: • authenticate_user_hashed
ignored_deletes	connection and table	The ignored_deletes parameter is available in the following events: • upload_statistics
ignored_inserts	connection and table	The ignored_inserts parameter is available in the following events: • upload_statistics
ignored_updates	connection and table	The ignored_updates parameter is available in the following events: • upload_statistics
inserted_rows	connection and table	The inserted_rows parameter is available in the following events: • upload_statistics
last_download	connection	The last_download parameter is available in the following events: • begin_download • end_download • modify_last_download_timestamp • modify_next_last_download_timestamp • nonblocking_download_ack • prepare_for_download

System parameter	Event type	Event parameter is available in
last_publication_download	connection	The last_publication_download parameter is available in the following events: • begin_publication • end_publication • publication_nonblocking_download_ack
last_publication_upload	connection	The last_publication_upload parameter is available in the following events: • begin_publication • end_publication
last_table_download	table	The last_table_download parameter is available in the following events: • begin_download • begin_download_deletes • begin_download_rows • download_cursor • download_delete_cursor • end_download • end_download_deletes • end_download_rows
maximum_time	connection and table	The maximum_time parameter is available in the following events: • time_statistics
minimum_time	connection and table	The minimum_time parameter is available in the following events: • time_statistics
new_password	connection	The new_password parameter is available in the following events: • authenticate_user
next_last_download	connection	The next_last_download parameter is available in the following events: • generate_next_last_download_timestamp • modify_next_last_download_timestamp

System parameter	Event type	Event parameter is available in
number_of_calls	connection and table	The number_of_calls parameter is available in the following events: • time_statistics
odbc_state	connection	The odbc_state parameter is available in the following events: • handle_odbc_error • report_odbc_error
password	connection	The password parameter is available in the following events: • authenticate_user
publication_name	connection	The publication_name parameter is available in the following events: • begin_publication • end_publication • publication_nonblocking_download_ack

System parameter	Event type	Event parameter is available in
remote_id	connection and table	The remote_id parameter is available in the following events: • authenticate_parameters (connection event) • authenticate_user (connection event) • authenticate_user_hashed (connection event) • begin_download (connection and table event) • begin_download_deletes (table event) • begin_download_rows (table event) • begin_publication (connection event) • begin_synchronization (connection and table event) • begin_upload (connection and table event) • begin_upload_deletes (table event) • begin_upload_rows (table event) • download_cursor (table event) • download_delete_cursor (table event) • download_statistics (connection and table event) • end_download (connection and table event) • end_download_deletes (table event) • end_download_rows (table event) • end_publication (connection event) • end_synchronization (connection and table event) • end_upload (connection and table event) • end_upload_deletes (table event) • end_upload_rows (table event) • generate_next_last_download_timestamp (connection) • handle_error (connection event) • handle_odbc_error (connection event) • modify_error_message (connection event) • modify_last_download_timestamp (connection event) • modify_next_last_download_timestamp (connection event) • modify_user (connection event) • nonblocking_download_ack (connection event) • prepare_for_download (connection event) • publication_nonblocking_download_ack (connection event) • report_error (connection event) • report_odbc_error (connection event) • resolve_conflict (table event)

System parameter	Event type	Event parameter is available in
		• synchronization_statistics (connection and table event) • time_statistics (connection and table event) • upload_delete (table event) • upload_fetch (table event) • upload_fetch_column_conflict (table event) • upload_insert (table event) • upload_new_row_insert (table event) • upload_old_row_insert (table event) • upload_statistics (connection and table event) • upload_update (table event)
remote_key	connection	The remote_key parameter is available in the following events: • authenticate_file_transfer • authenticate_file_upload
subdir	connection	The subdir parameter is available in the following events: • authenticate_file_transfer • authenticate_file_upload
subscription_id	connection	The subscription_id parameter is available in the following events: • begin_publication • end_publication • publication_nonblocking_download_ack
synchronization_ok	connection and table	The synchronization_ok parameter is available in the following events: • end_synchronization
synchronized_tables	connection	The synchronized_tables parameter is available in the following events: • synchronization_statistics

System parameter	Event type	Event parameter is available in
table	connection and table	The table parameter is available in the following events: • begin_download (table event) • begin_download_deletes (table event) • begin_download_rows (table event) • begin_synchronization (table event) • begin_upload (table event) • begin_upload_deletes (table event) • begin_upload_rows (table event) • download_statistics (table event) • end_download (table event) • end_download_deletes (table event) • end_download_rows (table event) • end_synchronization (table event) • end_upload (table event) • end_upload_deletes (table event) • end_upload_rows (table event) • handle_error (connection event) • handle_odbc_error (connection event) • report_error (connection event) • report_odbc_error (connection event) • resolve_conflict (table event) • synchronization_statistics (table event) • time_statistics (table event) • upload_statistics (table event)
total_time	connection and table	The total_time parameter is available in the following events: • time_statistics
updated_rows	connection and table	The updated_rows parameter is available in the following events: • upload_statistics

System parameter	Event type	Event parameter is available in
username	connection and table	The username parameter is available in the following events: • authenticate_file_transfer (connection event) • authenticate_file_upload (connection event) • authenticate_parameters (connection event) • authenticate_user (connection event) • authenticate_user_hashed (connection event) • begin_download (connection and table event) • begin_download_deletes (table event) • begin_download_rows (table event) • begin_publication (connection event) • begin_synchronization (connection and table event) • begin_upload (connection and table event) • begin_upload_deletes (table event) • begin_upload_rows (table event) • download_cursor (table event) • download_delete_cursor (table event) • download_statistics (connection and table event) • end_download (connection and table event) • end_download_deletes (table event) • end_download_rows (table event) • end_publication (connection event) • end_synchronization (connection and table event) • end_upload (connection and table event) • end_upload_deletes (table event) • end_upload_rows (table event) • generate_next_last_download_timestamp (connection event) • handle_error (connection event) • handle_odbc_error (connection error) • modify_error_message (connection error) • modify_last_download_timestamp (connection error) • modify_next_last_download_timestamp (connection event) • modify_user (connection event) • nonblocking_download_ack (connection event) • prepare_for_download (connection event) • publication_nonblocking_download_ack (connection event) • report_error (connection event)

System parameter	Event type	Event parameter is available in
		• report_odbc_error (connection event) • resolve_conflict (table event) • synchronization_statistics (connection and table event) • time_statistics (connection and table event) • upload_delete (table event) • upload_fetch (table event) • upload_fetch_column_conflict (table event) • upload_insert (table event) • upload_new_row_insert (table event) • upload_old_row_insert (table event) • upload_statistics (connection and table event) • upload_update (table event)
warnings	connection and table	The warnings parameter is available in the following events: • download_statistics • synchronization_statistics • upload_statistics

User-defined named parameters

You can also define your own parameters. These are especially useful for RDBMSs that don't allow user-defined variables.

User-defined parameters are defined (and set to null) when first referenced. They must start with ui and a period (ui.) if the parameter will only be referenced (input-only) and u and a period (u.) if the parameter will be updated (in/out). A user-defined parameter lasts for one synchronization—it is set to null at the start of every synchronization.

A typical use of user-defined parameters is to access state information without having to store it in a table (requiring potentially complex joins).

Example

For example, assume you create a stored procedure called MyCustomProc that sets a variable called var1 to custom_value:

```
CREATE PROCEDURE MyCustomProc(
  IN username VARCHAR(128), INOUT var1 VARCHAR(128)
)
begin
 SET var1 = 'custom_value';
end
```

The following begin_synchronization script defines the user-defined parameter var1 and sets the value to custom_value:


```
CALL ml_add_connection_script (
  'version1',
  'begin_synchronization',
  '{call MyCustomProc( {ml s.username}, {ml u.var1} )}' );
```

The following download_cursor script references var1, whose value is custom_value:

```
CALL ml_add_connection_script (
  'version1',
  'select pk,col1 from MyTable where u_name = {ml s.username} and
some_other_column = {ml ui.var1}' );
```

Assume you have another stored procedure called MyPFDProc that defines its first parameter to in/out. The following prepare_for_download script changes the value of var1 to pfd_value:

```
CALL ml_add_connection_script (
  'version1',
  'prepare_for_download',
  '{call MyPFDProc( {ml u.var1} )}' );
```

The following begin_download script references var1, whose value is now pfd_value:

```
CALL ml_add_connection_script (
  'version1',
  'begin_download',
  'insert into SomeTable values( {ml s.username}, {ml ui.var1} )' );
```

Authentication parameters

In MobiLink scripts, authentication parameters are named parameters that are prefaced with the letter a, such as {ml a.1}. The parameters must be numbers starting at 1, with a limit of 255. The values are sent up from MobiLink clients.

When used in the authenticate_* scripts, authentication parameters pass authentication information.

Authentication parameters can be used in all other events (except begin_connection and end_connection) to pass information from MobiLink clients. This technique is a convenient way to do something that you could otherwise do by uploading rows to a table. With authentication parameters the values are available prior to the table's upload events.

On SQL Anywhere remotes, you pass the information with the dbmlsync -ap option. On UltraLite remotes, you pass the information with auth_parms and num_auth_parms.

See also

- "Script parameters" on page 268
- dbmlsync: "-ap dbmlsync option" [*MobiLink - Client Administration*]
- UltraLite: "Authentication Parameters synchronization parameter" [*UltraLite - Database Management and Reference*] and "Number of Authentication Parameters parameter" [*UltraLite - Database Management and Reference*]

Example

For UltraLite remote databases, pass the parameters using the num_auth_parms and auth_parms fields in the ul_sync_info struct. num_auth_parms is a count of the number of parameters, from 0 to 255.

auth_parms is a pointer to an array of strings. During synchronization the authentication parameters are obfuscated in the same way as passwords. If num_auth_parms is 0, set auth_parms to null. The following is an example of passing parameters in UltraLite:

```
ul_char * Params[ 3 ] = { UL_TEXT( "param1" ),
   UL_TEXT( "param2" ), UL_TEXT( "param3" ) };

...
info.num_auth_parms = 3;
info.auth_parms = Params;
```

For SQL Anywhere remote databases, you pass authentication parameters using the dbmlsync -ap option, in a comma-separated list. For example, the following command line passes three parameters:

```
dbmlsync -ap "param1,param2,param3"
```

On the server, you reference the authentication parameters using the order in which they were sent up. In this example, the authenticate_parameters script could be:

```
CALL my_auth_parm (
  {ml s.authentication_status},
  {ml s.remote_id},
  {ml s.username},
  {ml a.1},
  {ml a.2},
  {ml a.3}
)
```

Script versions

Scripts are organized into groups called **script versions**. By specifying a particular script version, MobiLink clients can select which set of synchronization scripts are used to process the upload and prepare the download.

For information about how to add a script version to the consolidated database, see “Adding a script version” on page 283.

Application of script versions

Script versions allow you to organize your scripts into sets, which are run under different circumstances. This ability provides flexibility and is especially useful in the following circumstances:

- **Customizing applications** Using a different set of scripts to process information from different types of remote users. For example, you could write a different set of scripts for use when managers synchronize their databases than would be used for other people in the organization. Although you could achieve the same functionality with one set of scripts, these scripts would be more complicated.

- **Upgrading applications** When you want to upgrade a database application, new scripts may be needed because the new version of your application may handle data differently. New scripts are almost always necessary when the remote database changes. It is usually impossible to upgrade all users simultaneously. Since both old and new scripts can coexist on the server, all users can synchronize no matter which version of your application they are using.

- **Maintaining multiple applications** A single MobiLink server may need to synchronize two entirely different applications. For example, some employees may use a sales application, whereas others require an application designed for inventory control. When two applications require different sets of data, you can create two versions of the synchronization scripts, one script version for each application.

- **Setting properties for the script version** You can set properties for your script version that can be referenced from classes in .NET or Java synchronization logic. See "ml_add_property system procedure" on page 705.

Assigning script version names

A script version name is a string. You specify this name when you add a script to the consolidated database. For example, if you add your scripts with the ml_add_connection_script and the ml_add_table_script stored procedures, the script version name is the first parameter. Alternatively, if you add your scripts using Sybase Central, you are prompted for the script version name.

You cannot use the following names for script versions: **ml_sis_1** or **ml_qa_1**. These names are used internally by MobiLink.

Caution
It is strongly recommended that your script version names do not start with **ml_**. Script versions starting with ml_ are reserved for internal use.

Specifying a script version for a synchronization

If no script version is specified at the remote site when synchronization is initiated, the synchronization fails. See "Associating script versions with subscriptions" [*MobiLink - Client Administration*].

ml_global script version

You can create a script version called **ml_global** that is used differently from other script versions. If you create a script version called ml_global, you define it once and then the connection scripts associated with it are automatically used in all synchronizations. You never explicitly specify ml_global as a script version from a synchronization client.

If you define a script in the ml_global script version and then you define a script for the same event in the script version that you specify for the synchronization, the script from the specified script version is used. Scripts in the ml_global script version are only used if they are not defined in the primary script version that is being synchronized.

The ml_global script version can only contain connection-level scripts. It is optional, and may not be useful if you are using only one script version.

Adding a script version

All scripts are associated with a script version. When working in Sybase Central, you must add a script version name to your consolidated database before you can add any connection scripts. When adding scripts with system procedures, if you specify a new script version name it is automatically added with the

script. In Sybase Central, only one script version is allowed per synchronization model and it is by default given the same name as the synchronization model.

If you want to be able to perform schema changes without synchronizing, you must add a script version to the synchronization subscription using SQL syntax.

See "Script versions" on page 282.

To add a script version to a consolidated database (Sybase Central)

1. From the **View** menu, choose **Folders**.
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. Right-click the **Versions** folder and choose **File** » **New** » **Version**.
4. Follow the instructions in the **Create Script Version Wizard**.

To remove a script version from a consolidated database (Sybase Central)

1. From the **View** menu, choose **Folders**.
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. Under your consolidated database in the left pane, click **Versions**. A list of the script versions appears in the right pane.
4. In the right pane, right-click the script version you want to remove and select **Delete**.
5. Click **Yes**.

> **Note**
> All scripts associated with the script version are also deleted.

To add a script version to a consolidated database (system procedures)

- You can add a script version in the same operation as adding a connection script or table script. See "System procedures to add or delete scripts" on page 691.

Required scripts

When you run the MobiLink server, certain scripts are required. Which scripts are required is determined by whether you are doing a bi-directional, upload-only, or download-only synchronization.

For bi-directional or upload-only synchronization, MobiLink requires the following table scripts:

- upload_delete (if uploading deleted rows using SQL)
- upload_insert (if uploading inserted rows using SQL)
- upload_update (if uploading updated rows using SQL)
- Or, if you are processing the upload by direct row handling, MobiLink requires a script for the handle_UploadData connection event.

For bi-directional or download-only synchronization, MobiLink expects every table in the synchronization to have both a download_cursor and a download_delete_cursor. Or, if you are processing the download by direct row handling, MobiLink requires that you specify a handle_DownloadData connection script. Note that this script can be empty and you can process the download in any other event.

All required scripts must be specified. If a required script is missing the synchronization aborts. If there is a data script that you want ignored, use the prefix --{ml_ignore}. See "Ignoring scripts" on page 287.

Adding and deleting scripts

When you use the **Create Synchronization Model Wizard**, scripts are automatically added to the consolidated database when you deploy the model.

When you create synchronization scripts outside Sybase Central, you must add them to MobiLink system tables in the consolidated database. For SQL scripts, the entire script is saved in the MobiLink system table. For Java or .NET scripts, the method name is registered in the system table. The method for storing scripts and method names is similar.

See "MobiLink server system tables" on page 4.

If you are using Sybase Central, you must add a script version to the database before you can add individual scripts. See "Adding a script version" on page 283.

To add a connection script (Sybase Central)

1. From the **View** menu, choose **Folders**.
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. Right-click **Connection Scripts** and choose **New** » **Connection Script**.
4. Follow the instructions in the **Create Connection Script Wizard**.

To delete a connection script (Sybase Central)

1. From the **View** menu, choose **Folders**.

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. Expand **Connection Scripts**.

4. Right-click a connection script and choose **Delete**.

5. Click **Yes**.

To add a table script (Sybase Central)

1. From the **View** menu, choose **Folders**.

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. Expand **Synchronized Tables**.

4. Right-click the table and choose **New** » **Table Script**.

5. Follow the instructions in the **Create Table Script Wizard**.

To delete a table script (Sybase Central)

1. From the **View** menu, choose **Folders**.

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. Expand **Synchronized Tables**.

4. Expand the table.

5. Right-click the table script and choose **Delete**.

6. Click **Yes**.

To add or delete all types of scripts (system procedures)

- You can add scripts to a consolidated database or delete scripts from a consolidated database using stored procedures that are installed when you set up your consolidated database.

For a description of the stored procedures that you can use to add or delete scripts, see:

- "ml_add_connection_script system procedure" on page 694
- "ml_add_table_script system procedure" on page 708
- "ml_add_dnet_connection_script system procedure" on page 695
- "ml_add_dnet_table_script system procedure" on page 697
- "ml_add_java_connection_script system procedure" on page 698
- "ml_add_java_table_script system procedure" on page 699

Direct inserts of scripts

It is recommended that you use stored procedures or Sybase Central to insert scripts into the system tables. However, in some rare cases you may need to use an INSERT statement to directly insert the scripts. For example, versions of some RDBMSs may have length limitations that make it difficult to use stored procedures.

For a complete description of the MobiLink system tables, see "MobiLink server system tables" on page 4.

The format of the INSERT statements that are required to directly insert scripts can be found in the source code for the ml_add_connection_script and ml_add_table_script stored procedures. The source code for these stored procedures is located in the MobiLink setup scripts. There is a different setup script for each supported RDBMS. The setup scripts are all located in *install-dir\MobiLink\setup* and are called:

Consolidated database	Setup file
Adaptive Server Enterprise	*syncase.sql*
IBM DB2 LUW	*syncdb2.sql*
Microsoft SQL Server	*syncmss.sql*
MySQL	*syncmys.sql*
Oracle	*syncora.sql*
SQL Anywhere	*syncsa.sql*

Ignoring scripts

If an upload stream contains insert, update, or delete data for a table that has no upload_insert, upload_update, and upload_delete script in the consolidated database, or if there is no download script (download_cursor and download_delete_cursor scripts) for the table, then the MobiLink server complains about the missed scripts and aborts the synchronization.

The warning messages can be suppressed with the -zwd MobiLink server command option, however, this option suppresses the warning messages for all the synchronization tables.

Now, the MobiLink server treats any connection and table scripts that contain the prefix --{ml_ignore} differently. The MobiLink server recognizes these scripts as intentionally ignored scripts. More precisely, if an upload stream contains insert, update, or delete data for a synchronization table that has an upload_insert, upload_update, or upload_delete script with the prefix --{ml_ignore}, the MobiLink server does not execute these scripts against the consolidated database and continues the synchronization without showing any error or warning messages. The uploaded rows are ignored.

When a table is downloaded, both the download_cursor and download_delete_cursor scripts must be defined. To prevent downloading rows, define either or both of these scripts as --{ml_ignore}, as required.

Writing scripts to upload rows

To inform the MobiLink server on how to process the upload data received from the remote databases, you define upload scripts. You write separate scripts to handle rows that are updated, inserted, or deleted at the remote database. A simple implementation would perform corresponding actions (update, insert, delete) at the consolidated database.

The MobiLink server uploads data in a single transaction. For a description of the upload process, see "Events during upload" on page 303.

For techniques for uploading rows using Java or .NET synchronization logic, see "Uploading or downloading rows" on page 597.

Notes

- The begin_upload and end_upload scripts for each remote table hold logic that is independent of the individual rows being updated.
- The upload consists of single row inserts, updates, and deletes. These actions are typically performed using upload_insert, upload_update, and upload_delete scripts.
- To prepare the upload for SQL Anywhere clients, the dbmlsync utility requires access to all transaction logs written since the last successful synchronization. See "Transaction log files" [*MobiLink - Client Administration*].

Writing upload_insert scripts

The MobiLink server uses this event during processing of the upload to handle rows inserted into the remote database.

The following is an INSERT statement used in an upload_insert script.

```
INSERT INTO emp ( emp_id, emp_name )
VALUES ( { ml r.emp_id }, { ml r.emp_name } );
```


Notes

- When using question marks instead of named parameters as placeholders, the upload_new_row_insert and upload_old_row_insert events accept remote_id and user_name as extra parameters. These parameters must appear before the full column list of the table.

See also

- "Writing scripts to upload rows" on page 288
- "upload_insert table event" on page 487

Writing upload_update scripts

The MobiLink server uses this event during processing of the upload to handle rows updated at the remote database. The following UPDATE statement could be used as an upload_update script for the emp table.

```
UPDATE emp
SET emp_name = {ml r.emp_name}
WHERE emp_id = {ml o.emp_id};
```

Notes

- When using question marks instead of named parameters as placeholders, the number of parameters can be equal to one of the following (the use of question marks in SQL scripts has been deprecated):
  - The number of non-primary key columns + primary key columns.
  - 2 * (the number of non-primary key columns + primary key columns).

  The column order must consist of non-primary key columns first, followed by one of the following:
  - The primary key columns.
  - All the columns.

See also

- "Writing scripts to upload rows" on page 288
- "upload_update table event" on page 519

Writing upload_delete scripts

The MobiLink server uses this event during processing of the upload to handle rows deleted from the remote database. The following statement shows how to use the upload_delete statement.

```
DELETE FROM emp
WHERE emp_id = {ml r.emp_id};
```

Notes

- When using question marks instead of named parameters as placeholders, the number of parameters must be equal to one of the following (the use of question marks in SQL scripts has been deprecated):

- The number of primary key columns.
- The number of all columns.

See also

- "Writing scripts to upload rows" on page 288
- "upload_delete table event" on page 436

Writing upload_fetch scripts

The upload_fetch script is a SELECT statement that defines a cursor in the consolidated database table. This cursor is used to compare the old values of updated rows, as received from the remote database, against the current value in the consolidated database. In this way, the upload_fetch script identifies conflicts when updates are being processed.

Given a synchronized table defined as:

```
CREATE TABLE uf_example (
 pk1 integer NOT NULL,
 pk2  integer NOT NULL,
 val  varchar(200),
 PRIMARY KEY( pk1, pk2 ));
```

Then one possible upload_fetch script for this table is:

```
SELECT pk1, pk2, val
FROM uf_example
WHERE pk1 = {ml r.pk1} and pk2 = {ml r.pk2}
```

See "upload_fetch table event" on page 454.

The MobiLink server requires the WHERE clause of the query in the upload_fetch script to identify exactly one row in the consolidated database to be checked for conflicts.

Writing scripts to download rows

There are two scripts that can be used for processing each table during the download transaction. These are the download_cursor script, which performs inserts and updates, and the download_delete_cursor script, which performs deletes.

These scripts are either SELECT statements or calls to procedures that return result sets. The MobiLink server downloads the result set of the script to the remote database. The MobiLink client automatically inserts or updates rows based on the download_cursor script result set, and deletes rows based on the download_delete_cursor event.

For more information about using stored procedures, see "Downloading a result set from a stored procedure call" on page 117.

The MobiLink server downloads data in a single transaction. For a description of the download process, see "Events during download" on page 307.

Notes

- Like the upload, the download starts and ends with connection events. Other events are table-level events.
- The begin_download and end_download scripts for each remote table hold logic that is independent of the individual rows being updated.
- The download does not distinguish between inserts and updates. The script associated with the download_cursor event is a SELECT statement that defines the rows to be downloaded. The client detects whether the row exists or not and then it performs the appropriate insert or update operation.
- For timestamp-based downloads, you specify the last_table_download parameter to ensure that only changes since the last synchronization are downloaded. For example, the download_cursor or download_delete_cursor SQL script could include the line:

  ```
  WHERE Customer.last_modified >= {ml s.last_table_download}
  ```

  See "Using last download times in scripts" on page 88.
- At the end of the download processing, the client automatically deletes rows if necessary to avoid referential integrity violations.
- If you change the SendDownloadAck setting to ON, the download transaction is committed but the acknowledgement scripts are not executed until the acknowledgement is received.

  By default, SendDownloadAck is set to OFF.

  See "SendDownloadAck (sa) extended option" [*MobiLink - Client Administration*], "Send Download Acknowledgement synchronization parameter" [*UltraLite - Database Management and Reference*], "nonblocking_download_ack connection event" on page 410 and "publication_nonblocking_download_ack connection event" on page 414.

Caution
Do not synchronize shadow tables that were created by previous deployments (for example, tables ending with *_mod* or *_del* should not be synchronized). These tables are only needed by the consolidated database to track modified or deleted rows.

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

Writing download_cursor scripts

You write download_cursor scripts to download rows from the consolidated database to your remote database. Similarly, you write download_delete_cursor scripts to download rows to delete from the remote database. You must write both of these scripts for each table in the remote database for which you want to download changes. You can use other scripts to customize the download process, but no others are necessary.

- Each download_cursor script that you want to download rows must contain a SELECT statement or a call to a procedure that contains a SELECT statement.

- If you do not want download rows, define the script as --{ml_ignore}. Alternatively, you can use the ml_add_missing_dnld_scripts system procedure to define missing download scripts as ignored. See "ml_add_missing_dnld_scripts system procedure" on page 700.
- The download_cursor script must select all columns that correspond to the columns in the corresponding table in the remote database. The columns in the consolidated database can have different names than the corresponding columns in the remote database, but they must be of compatible types.

Example

The following script could serve as a download_cursor script for a remote table that holds employee information. This script downloads information about all the employees.

```
SELECT emp_id, emp_fname, emp_lname
FROM employee;
```

The MobiLink server passes specific parameters to some scripts. The MobiLink server substitutes the value of the parameter before executing the statement against the consolidated database. The use of question marks has been deprecated in SQL scripts. The following script shows how you can use named parameters:

```
CALL ml_add_table_script(
   'Lab',
   'ULOrder',
   'download_cursor',
   'SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc, o.quant,
o.notes, o.status
         FROM ULOrder o
         WHERE o.last_modified >= {ml s.last_table_download}
               AND o.emp_name = {ml s.username}' )
```

Notes

- All cursor scripts must select the columns in the same order as the columns are defined in the remote database. Where column names or table structure is different in the consolidated database, columns should be selected in the correct order for the remote database, or equivalently, the reference database. Columns are assigned to columns in the remote database based on their order in the SELECT statement.
- Row values can be selected from a single table or from a join of multiple tables.
- The remote table need not have the same name as the table in the consolidated database. The script itself need not include the name of the remote table. The name of the remote table is identified by an entry in the ml_table MobiLink system table. Use Sybase Central to view the remote tables listed together with their scripts.

See also

- "download_cursor table event" on page 347
- "Partitioning rows among remote databases" on page 92
- "Writing download_delete_cursor scripts" on page 293

Writing download_delete_cursor scripts

You write download_delete_cursor scripts to delete rows from your remote database. You must write one of these scripts for each table in the remote database participating in the download. If you do not want to delete rows, define each script as --{ml_ignore}. Alternatively, you can use the ml_add_missing_dnld_scripts system procedure to define missing download scripts as ignored. See "ml_add_missing_dnld_scripts system procedure" on page 700.

You cannot just delete rows from the consolidated database and have them disappear from remote databases. You need to keep track of the primary keys for deleted rows, so that you can select those primary keys with your download_delete_cursor. There are two common techniques for achieving this:

- **Logical deletes** Do not physically delete the row in the consolidated database. Instead, have a status column that keeps track of whether rows are valid. This simplifies the download_delete_cursor. However, the download_cursor and other applications may need to be modified to recognize and use the status column. If you have a last modified column that holds the time of deletion, and if you also keep track of the last download time for each remote, then you can physically delete the row once all remote download times are newer than the time of deletion.

- **Shadow table** For each table for which you want to track deletes, create a shadow table with two columns, one holding the primary key for the table, and the other holding a timestamp. Create a trigger that inserts the primary key and timestamp into the shadow table whenever a row is deleted. Your download_delete_cursor can then select from this shadow table. As with logical deletes, you can delete the row from the shadow table once all remote databases have downloaded the corresponding data.

The MobiLink server deletes rows in the remote database by selecting primary key values from the consolidated database and passing those values to the remote database. If the values match those of a primary key in the remote database, then that row is deleted.

- Each download_delete_cursor script that you want to download deletes must contain a SELECT statement or a call to a stored procedure that returns a result set. The MobiLink server uses this statement to define a cursor in the consolidated database.

- If you always want a download_delete_cursor to select no rows, define the script as --{ml_ignore}.

- This statement must select all the columns that correspond to the primary key columns in the table in the remote database. The columns in the consolidated database can have different names than the corresponding columns in the remote database, but they must be of compatible types.

- The values must be selected in the same order as the corresponding columns are defined in the remote database. That order is the order of the columns in the CREATE TABLE statement used to make the table, not the order they appear in the statement that defines the primary key.

- If you delete a parent record at the remote via a download_delete_cursor, the child records are automatically deleted as well.

  For more information about deleting child records, see "Referential integrity and synchronization" [*MobiLink - Getting Started*].

Deleting all the rows in a table

When MobiLink detects a download_delete_cursor with a row that contains all nulls, it deletes all the data in the remote table. The number of nulls in the download_delete_cursor can be the number of primary key columns or the total number of columns in the table.

For example, the following download_delete_cursor SQL script deletes every row in a table in which there are two primary key columns. This example works for SQL Anywhere, Adaptive Server Enterprise, and Microsoft SQL Server databases.

```
SELECT NULL, NULL
```

In IBM DB2 LUW and Oracle consolidated databases, you must specify a dummy table to select null. For IBM DB2 LUW 9.5 and 9.7, you can use the following syntax:

```
SELECT CAST( NULL AS INTEGER ), CAST( NULL AS INTEGER ) FROM SYSIBM.SYSDUMMY1
```

For Oracle consolidated databases, you can use the following syntax:

```
SELECT NULL, NULL FROM DUAL
```

Examples

The following example is a download_delete_cursor script for a remote table that holds employee information. The MobiLink server uses this SQL statement to define the delete cursor. This script deletes information about all orders that are both in the consolidated and remote databases at the time the script is executed.

```
SELECT order_id
FROM ULorder
```

The download_delete_cursor accepts the parameters last_table_download and username. The following script shows how you can use each parameter to narrow your selection.

```
SELECT order_id
FROM ULOrder
WHERE last_modified >= {ml s.last_table_download}
 AND status = 'Approved'
 AND user_name = {ml s.username}
```

Another strategy is to allow the client application to delete the rows itself. This method is possible only when a rule identifies the unnecessary rows. For example, rows might contain a timestamp that indicates an expiry date. Before you delete the rows at the remote, use the STOP SYNCHRONIZATION DELETE statement to stop these deletes being uploaded during the next synchronization. Be sure to execute START SYNCHRONIZATION DELETE immediately afterward if you want other deletes to be synchronized in the normal fashion.

Notes

- The download_delete_cursor script must contain primary key columns in the same order as they are defined in the remote database.

- You can use the referential integrity checking built into all MobiLink clients to delete rows in an efficient manner by deleting only the parent rows. See "Referential integrity and synchronization" [*MobiLink - Getting Started*].

See also

For more information about using download_delete_cursor scripts, see:

- "download_cursor table event" on page 347
- "download_delete_cursor table event" on page 349
- "Handling deletes" on page 112
- "Temporarily stopping the synchronization of deletes" on page 113
- "STOP SYNCHRONIZATION DELETE statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "Partitioning rows among remote databases" on page 92
- "Snapshot synchronization" on page 90

Writing scripts to handle errors

An error in a synchronization script occurs when an operation in the script fails while the MobiLink server is executing it. For SQL scripts, the DBMS returns a SQLCODE and error message to the MobiLink server indicating the nature of the error. Each consolidated database DBMS has its own set of SQLCODEs and messages. By default, the MobiLink server rolls back the transaction in the consolidated database, logs the error, and aborts the synchronization.

When an error occurs during the invocation of a SQL data script, the MobiLink server invokes the handle_error or handle_odbc_error events. When these error-handling scripts are defined, the MobiLink server invokes them and passes several parameters providing information about the nature and context of the error. One parameter is an output value, called the action_code, to tell MobiLink server how to respond to the error. The action_code tells the MobiLink server to either ignore the error or abort the synchronization.

The error-handling scripts do **not** get invoked for all SQL errors. Only data scripts cause the error-handling scripts to be invoked. When errors occur in non-data scripts, the MobiLink server rolls back the transaction in the consolidated database, logs the error, and aborts the synchronization.

If your consolidated DBMS supports exception handling, consider using it instead of the error-handling scripts—particularly if you need to ignore certain errors in data scripts. Using exception handling will almost always perform better than the error-handling scripts.

If the handle_error or handle_odbc_error script itself causes an error, the MobiLink server rolls back the transaction in the consolidated database, logs the error, and aborts the synchronization.

Error handling actions

Some actions you may want to take in an error-handling script are:

- Ignore the error, but log it in an audit table.

- Instruct the MobiLink server to rollback the synchronization.
- Send an email alert message.

Handling multiple errors in a single SQL statement

ODBC allows multiple errors per SQL statement, and some RDBMSs make use of this feature. Microsoft SQL Server, for example, can have two errors for a single statement. The first is the actual error, and the second is usually an informational message telling you why the current statement has been terminated.

When a single SQL statement causes multiple errors, the handle_error script is invoked once per error. The MobiLink server uses the most severe action code (that is, the numerically greatest) to determine the action to take. The same applies to the handle_error script.

If the handle_error script itself causes a SQL error, then the default action code (3000) is assumed.

See:

- "handle_error connection event" on page 387
- "handle_odbc_error connection event" on page 391
- "report_error connection event" on page 416
- "report_odbc_error connection event" on page 419

Reporting errors

Since errors cause a rollback in the consolidated database by default, it is difficult to create a log of errors and their resolutions within the consolidated database due to the rollback. The report_error and report_odbc_error events let you create a proper record of user-defined script errors because they are invoked on a different database connection than the synchronization. These error-reporting scripts are invoked immediately after the error-handling scripts are invoked, and are immediately followed by a commit.

The error-reporting scripts get invoked for all SQL errors that occur in user defined scripts. When errors occur in user-defined scripts, the MobiLink server rolls back the transaction in the consolidated database, logs the error, and aborts the synchronization.

If your consolidated DBMS supports an out-of-band (outside of the current database connection) mechanism for reporting activity from SQL, consider using that mechanism instead of the error-reporting scripts defined by MobiLink.

See:

- "handle_error connection event" on page 387
- "handle_odbc_error connection event" on page 391
- "report_error connection event" on page 416
- "report_odbc_error connection event" on page 419

Example

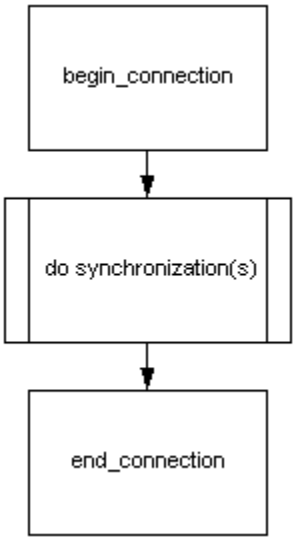
The following report_error script, which consists of a single insert statement, adds the script parameters into a table, along with the current date and time. The script does not commit this change because the MobiLink server always does so automatically.

```
INSERT INTO errors
VALUES(
  CURRENT DATE,
  {ml s.action_code},
  {ml s.error_code},
  {ml s.error_message},
  {ml s.username},
  {ml s.table} );
```

Synchronization events

Overview of MobiLink events

When a synchronization request occurs and the MobiLink server decides that a new consolidated database connection must be created, the begin_connection event is fired and synchronization starts.



Following the synchronization, the consolidated database connection is placed in a connection pool, and MobiLink again waits for a synchronization request. If another synchronization request for the same version is received, then MobiLink handles the next synchronization request on the same connection. Before a connection is eventually dropped from the connection pool, the end_connection event is fired.

There are many events in each synchronization. Most events are organized by the transaction containing them.

Transactions

Within each synchronization, the following transactions may occur.

- authentication
- begin synchronization
- upload
- prepare for download
- download
- end synchronization
- non-blocking download acknowledgement

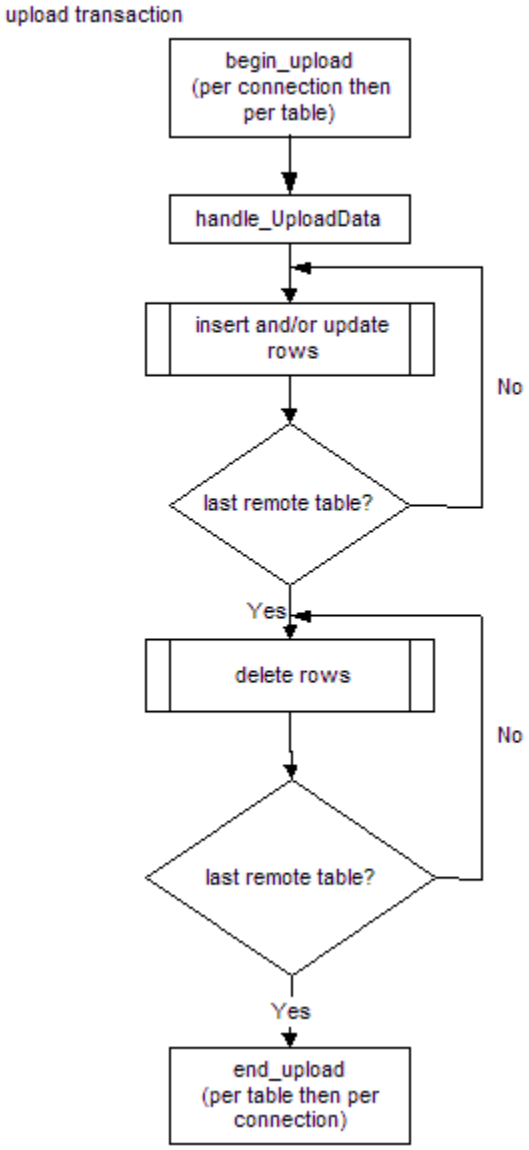
In addition, you can have two connection transactions. A begin connection transaction occurs right after a consolidated database connection is made, and an end connection transaction occurs when the connection is closed.

The primary phases of a synchronization are the upload and download transactions. The events contained in the upload and download transactions are outlined below.

The upload transaction

The upload transaction applies changes uploaded from a remote database.

The begin_upload event marks the beginning of the upload transaction. The upload transaction is a two-part process. First, inserts and updates are uploaded for all remote tables, and second, deletes are uploaded for all remote tables.



The end_upload event marks the end of the upload transaction.

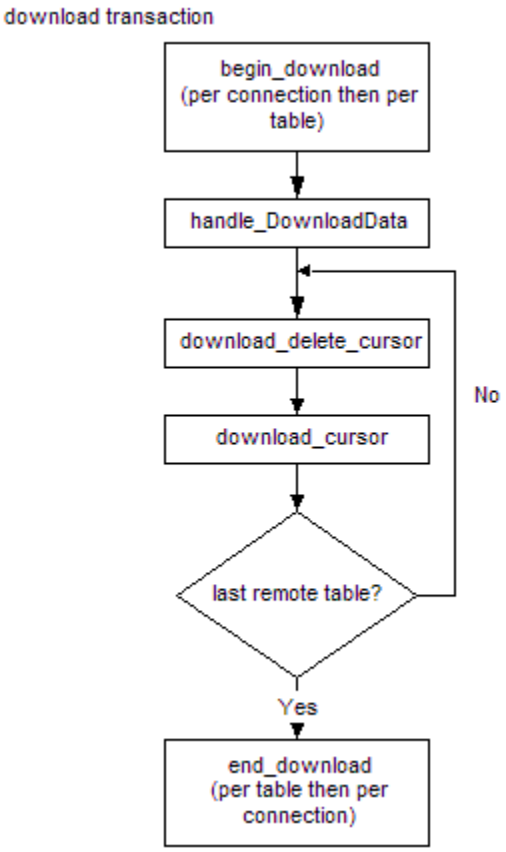
You can specify multiple upload transactions with the dbmlsync -tu option.

See "Writing scripts to upload rows" on page 288.

The download transaction

The download transaction fetches rows from the consolidated database. It begins with the begin_download event.

The download transaction is a two-part process. For each table, first deletes are downloaded, and then update/insert rows (upserts) are downloaded. The end_download event ends the download transaction.



See "Writing scripts to download rows" on page 290.

The non-blocking download acknowledgement transaction

The non-blocking download acknowledgement transaction is only performed when a download acknowledgement is received. This transaction has two purposes. The scripts publication_nonblocking_download_ack and nonblocking_download_ack are run in this transaction; they help download status tracking. Secondly, download timestamps in the MobiLink system tables are updated during this transaction.

Note that this transaction may not be performed on the same database connection as the other events for the target synchronization. This means that no connection level variables may be referenced in this transaction.

Event overview in pseudocode

The following pseudocode provides an overview of the sequence in which events, and the scripts of the same names, are invoked. This representation of the MobiLink event model assumes a full synchronization (not upload-only or download-only) with no errors.

Notes

- Usually, if you have not defined a script for a given event, the default action is to do nothing.
- The begin_connection and end_connection events are **connection-level events**. They are independent of any single synchronization and have no parameters.
- Some events are invoked once per synchronization for each table being synchronized. Scripts associated with these events are called **table-level scripts**.

  While each table can have its own table scripts, you can also write table-level scripts that are shared by several tables.
- Some events, such as begin_synchronization, occur at both the connection level and the table level. You can supply both connection and table scripts for these events.
- The COMMIT statements illustrate how the synchronization process is broken up into distinct transactions.

Caution
There should be no implicit or explicit commit or rollback in your SQL synchronization scripts or the procedures or triggers that are called from your SQL synchronization scripts. COMMIT or ROLLBACK statements within SQL scripts alter the transactional nature of the synchronization steps. If you use them, MobiLink cannot guarantee the integrity of your data in the event of a failure.

MobiLink complete event model

```
------------------------------------------------------
MobiLink complete event model.

Legend:
- // This is a comment.
- <name>
    The pseudo code for <name> is listed separately
    in a later section, under a banner:
        ------------------------
        name
        ------------------------
- VariableName <- value
    Assign the given value to the given variable name.
    Variable names are in mixed case.
- event_name
    If you have defined a script for the given event name,
    it is invoked.
------------------------------------------------------

CONNECT to consolidated database
begin_connection_autocommit
begin_connection
COMMIT
for each synchronization request with
```

```
     the same script version {
  <synchronize>
}
end_connection
COMMIT
DISCONNECT from consolidated database

-----------------------------------------------------
synchronize
-----------------------------------------------------

<authenticate>
<begin_synchronization>
<upload>
<prepare_for_download>
<download>
<end_synchronization>

-----------------------------------------------------
authenticate
-----------------------------------------------------

Status <- 1000
UseDefaultAuthentication <- TRUE
if( authenticate_user script is defined ) {
  UseDefaultAuthentication <- FALSE
  TempStatus <- authenticate_user
  if( TempStatus > Status ) {
    Status <- TempStatus
  }
}
if( authenticate_user_hashed script is defined ) {
  UseDefaultAuthentication <- FALSE
  TempStatus <- authenticate_user_hashed
  if( TempStatus > Status ) {
    Status <- TempStatus
  }
}
  if( authenticate_parameters script is defined )
 {
    TempStatus <- authenticate_parameters
    if( TempStatus > Status ) {
      Status <- TempStatus
  }
if( UseDefaultAuthentication ) {
  if( the user exists in the ml_user table ) {
    if( ml_user.hashed_password column is not NULL ) {
      if( password matches ml_user.hashed_password ) {
        Status <- 1000
      } else {
        Status <- 4000
      }
    } else {
      Status <- 1000
    }
  } else if( -zu+ was on the command line ) {
    Status <- 1000
  } else {
    Status <- 4000
  }
}
if( Status >= 3000 ) {
  // Abort the synchronization.
} else {
```



```
  // UserName defaults to MobiLink user name
  // sent from the remote.
  if( modify_user script is defined ) {
    UserName <- modify_user
    // The new value of UserName is later passed to
    // all scripts that expect the MobiLink user name.
  }
}
COMMIT

-------------------------------------------------------
begin_synchronization
-------------------------------------------------------

begin_synchronization   // Connection event.
for each table being synchronized {
    begin_synchronization    // Call the table level script.
}
for each publication being synchronized {
  begin_publication
}
COMMIT

-------------------------------------------------------
end_synchronization
-------------------------------------------------------

for each publication being synchronized {
  if( begin_publication script was processed ) {
    end_publication
  }
}
for each table being synchronized {
  if( begin_synchronization table script was processed ) {
    end_synchronization // Table event.
  }
}
if( begin_synchronization connection script was processed ) {
  end_synchronization     // Connection event.
}
for each table being synchronized {
synchronization_statistics // Table event.
}
synchronization_statistics // Connection event.
for each table being synchronized {
  time_statistics // Table event.
}
time_statistics // Connection event.

COMMIT
```

For the details of upload processing, see "Events during upload" on page 303.

For the details of download processing, see "Events during download" on page 307.

Events during upload

The following pseudocode illustrates how upload events and upload scripts are invoked. The pseudocode uses the following conventions:

- **A script is real** This means the script is defined as a real script that will be executed against the consolidated database during synchronization.
- **A script is defined as an ignored script** This means the script is defined as an ignored script using "--{ml_ignore}".
- **A script is defined** This means the script is defined as a real or an ignored script.
- **A script is not defined** This means there is no script defined for the event at all. You must ignore a script if it is a required script, but you do not want to use that script.

These events take place at the upload location in the complete event model. See "Overview of MobiLink events" on page 297.

Overview of the upload

```
------------------------------------------------------
upload
------------------------------------------------------

begin_upload // Connection event
for each table being synchronized {
  begin_upload // Table event
}
  handle_UploadData
  for each table being synchronized {
    begin_upload_rows
    for each uploaded INSERT or UPDATE for this table {
      if( INSERT ) {
        <upload_inserted_row>
      }
      if( UPDATE ) {
        <upload_updated_row>
      }
    }
    end_upload_rows
  }
  for each table being synchronized IN REVERSE ORDER {
    begin_upload_deletes
    for each uploaded DELETE for this table {
      <upload_deleted_row>
    }
    end_upload_deletes
  }

For each table being synchronized {
  if( begin_upload table script was processed ) {
    end_upload // Table event
  }
}
if( begin_upload connection script was processed ) {
  end_upload // Connection event

  for each table being synchronized {
    upload_statistics  // Table event.
  }
    upload_statistics  // Connection event.

  COMMIT
```


Upload inserts

```
------------------------------------------------------
<upload_inserted_row>
------------------------------------------------------
// NOTES:
// - Only table scripts for the current table are involved.

 ConflictsAreExpected <- (
    upload_new_row_insert script is defined or
    upload_old_row_insert script is defined )
if( upload_insert script is real ) {
    upload_insert
} else if( ConflictsAreExpected and
    upload_update script is not defined and
    upload_insert script is not defined and
    upload_delete script is not defined ) {
    // Forced conflict which is deprecated
    if( upload_new_row_insert script is real ) {
        upload_new_row_insert
        resolve_conflict
    } else if( upload_new_row_insert is defined as an ignored script ) {
        // Ignore this insert
    } else {
        error
    }
} else if( handle_uploadData script is real or
    upload_insert script is defined as an ignored script ) {
    // Ignore the insert. (Only ignored in SQL, possibly handled by
handle_uploadData.)
} else {
    error
}
```

Upload updates

```
------------------------------------------------------
upload_updated_row
------------------------------------------------------
// NOTES:
// - Only table scripts for the current table are involved.
// - Both the old (original) and new rows are uploaded for
//   each update.

 ConflictsAreExpected <- (
   upload_new_row_insert script is defined or
   upload_old_row_insert script is defined )
Conflicted <- FALSE
if( upload_update script is real ) {
    if( upload_fetch script is real ) {
        if( ConflictsAreExpected ) {
            FETCH using upload_fetch INTO current_row
            if( current_row <> old row ) {
                Conflicted <- TRUE
            } else {
                upload_update
            }
        } else {
            error
        }
    } else if( upload_fetch script is not defined ) {
        if( ConflictsAreExpected ) {
            // This conflict detection approach is deprecated
            upload_update
```

```
            if( number of affected rows is 0 ) {
                Conflicted <- TRUE
            }
        } else {
            // No conflict detection and resolution by the MobiLink server
            // The upload_update script should handle conflict detection and
resolution
            upload_update
        }
    } else {
    // the upload_ftech script cannot defined as an ignored script
    error
    }
} else if( ConflictsAreExpected and
    upload_update script is not defined and
    upload_insert script is not defined and
    upload_delete script is not defined ) {
    // Forced conflict which is deprecated
    Conflicted <- TRUE
} else if( handle_uploadData script is defined or upload_update script is
defined as an ignored script ) {
    // Ignore the upload update (Only ignored in SQL, possibly handled by
handle_uploadData.)
} else {
    error
}
if( Conflicted ) {
    if( upload_old_row_insert script is real ) {
        upload_old_row_insert
    } else if( upload_old_row_insert script is defined as ignored script ) {
        // Ignore the old value
    } else {
        error
    }
    if( upload_new_row_insert script is real ) {
        upload_new_row_insert
    } else if( upload_new_row_insert script is defined as ignored script ) {
        // Ignore the new value
    } else {
        error
    }
    if( no error ) {
        resolve_conflict
    }
}
```

Upload deletes

```
-------------------------------------------------------
upload_deleted_row
-------------------------------------------------------
// NOTES:
// - Only table scripts for the current table are involved.

  ConflictsAreExpected <- (
    upload_new_row_insert script is defined or
    upload_old_row_insert script is defined )
if( upload_delete is real ) {
    upload_delete
} else if( ConflictsAreExpected
    and upload_update script is not defined
    and upload_insert script is not defined
    and upload_delete script is not defined ) {
    // Forced conflict which is deprecated
```



```
    if( upload_old_row_insert is real ) {
        upload_old_row_insert
        resolve_conflict
    } else if( upload_old_row_insert is defined as an ignored script ) {
        // Ignore the delete
    } else {
        error
    }
} else if( handle_UploadData script is real or
    upload_delete script is defined as an ignored script ) {
    // Ignore this delete. (Only ignored in SQL, possibly handled by
handle_uploadData.)
} else {
    error
}
```

Events during download

The following pseudocode provides an overview of the sequence in which download events, and the script of the same name, are invoked.

These events take place at the download location in the complete event model provided in “Overview of MobiLink events” on page 297.

```
-------------------------------------------------------
prepare_for_download
-------------------------------------------------------

generate_next_last_download_timestamp
modify_last_download_timestamp
fetch the next download timestamp from consolidated
prepare_for_download

-------------------------------------------------------
download
-------------------------------------------------------

begin_download // Connection event.
for each table being synchronized {
   begin_download // Table event.
}
   handle_DownloadData
   for each table being synchronized {
     begin_download_deletes
     for each row in download_delete_cursor {
       if( all primary key columns are NULL ) {
         send TRUNCATE to remote
       } else {
         send DELETE to remote
       }
     }
     end_download_deletes
     begin_download_rows
     for each row in download_cursor {
       send INSERT ON EXISTING UPDATE to remote
     }
     end_download_rows
   }
   modify_next_last_download_timestamp
   for each table being synchronized {
```

```
      if( begin_download table script was processed ) {
         end_download // Table event
      }
}
if( begin_download connect script was processed ) {
   end_download // Connection event
}
   for each table being synchronized {
     download_statistics   // Table event.
   }
     download_statistics   // Connection event.

COMMIT
```

Notes

- The download stream does not distinguish between inserts and updates. The script associated with the download_cursor event is a SELECT statement that defines the rows to be downloaded. The client detects whether the row exists and then it performs the appropriate insert or update operation.
- At the end of the download processing, the client automatically deletes rows that violate referential integrity.

  See “Referential integrity and synchronization” [*MobiLink - Getting Started*].

Data scripts

Scripts that directly handle row data are called data scripts. All other scripts are non-data scripts. The distinction between a data script and a non-data script is sometimes important. For example, only the data scripts can reference the named parameters for column values.

The following events have data scripts associated with them:

- “download_cursor table event” on page 347
- “download_delete_cursor table event” on page 349
- “handle_UploadData connection event” on page 394
- “handle_DownloadData connection event” on page 383
- “upload_delete table event” on page 436
- “upload_fetch table event” on page 454
- “upload_fetch_column_conflict table event” on page 469
- “upload_insert table event” on page 487
- “upload_new_row_insert table event” on page 505
- “upload_old_row_insert table event” on page 507

- “upload_update table event” on page 519

Java and .NET data scripts returning SQL (deprecated)

Starting in version 12, the ability for Java and .NET scripting logic to return strings that are interpreted by MobiLink server as SQL scripts is deprecated in data scripts. If your scripts need to cause changes in the consolidated database, they should do so directly from Java or .NET.

Following is an example of how a script could be updated. The first example uses SQL and the second example does not.

```
public String beginDownloadConnection(
  Timestamp ts,
  String user )
  throws java.sql.SQLException
{
    doSomeWork( ts, user );
    return( "CALL do_some_sql( {ml s.last_download}, {ml s.username} )" );
}

public void beginDownloadConnection(
  Timestamp ts,
  String user )
  throws java.sql.SQLException
{
    doSomeWork( ts, user );

    Connection conn = DBConnectionContext.getConnection();
    PreparedStatment stmt = conn.prepareStatement( "CALL
do_some_sql( ?,? )" );
    stmt.setTimestamp( 1, ts );
    stmt.setString( 2, user );
    stmt.executeUpdate();
}
```

authenticate_file_transfer connection event

Implements custom authentication for file transfers using the mlfiletransfer utility or the MLFileDownload method.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.file_authentication_code	INTEGER. Required. This is an INOUT parameter. It indicates the overall success of the authentication. If this value is 1000-1999, file transfer is allowed. If this value is 2000-2999, file transfer is not allowed.	1
s.filename	VARCHAR(128). Required. This INOUT parameter is the name of the file that is being transferred that is to be authenticated. Do not include a path and do not use ellipsis (three dots), comma, forward slash (/) or backslash (\). The file must be located in the root transfer directory that you specified with the mlsrv12 -ftr or -ftru option, or in one of the subdirectories that are automatically created. If this is not set explicitly, the default is the filename that was passed to the MobiLink server by the client.	2
s.username	VARCHAR(128). The MobiLink user name.	3
s.subdir	VARCHAR(128). Required. This optional INOUT parameter sets the subdirectory location for the files to be transferred. To use the root directory, set this option to null. This option must not include ellipsis (three dots), comma, forward slash (/) or backslash (\). This defaults to remote_key if it is not set explicitly.	Not applicable
s.remote_key	VARCHAR(128). Optional IN parameter to specify a remote key for the file transfer.	Not applicable

Remarks

The MobiLink server executes this event before allowing any download file transfer using the mlfiletransfer utility or MLFileDownload method. It is executed after the user has authenticated using regular authentication. If this script is not defined, the file transfer is allowed.

The MLFileDownload method can only be used by UltraLite clients.

See also

- "Adding and deleting scripts" on page 285
- "-ftr mlsrv12 option" on page 50
- "-ftru mlsrv12 option" on page 50
- "MobiLink File Transfer utility (mlfiletransfer)" [*MobiLink - Client Administration*]
- "MLFileDownload method" [*UltraLite - C and C++ Programming*]
- UltraLite: "Using MobiLink file transfers" [*UltraLite - Database Management and Reference*]
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

authenticate_file_upload connection event

Implements custom authentication for file transfers using the mlfiletransfer utility or the MLFileUpload method.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.file_authentication_code	INTEGER. Required. This is an INOUT parameter. It indicates the overall success of the authentication. If this value is 1000-1999, file transfer is allowed. If this value is 2000-2999, file transfer is not allowed.	1
s.filename	VARCHAR(128). Required. This INOUT parameter is the name of the file that is being transferred that is to be authenticated. Do not include a path and do not use ellipsis (three dots), comma, forward slash (/) or backslash (\). The file must be located in the root transfer directory that you specified with the mlsrv12 -ftr or -ftru option, or in one of the subdirectories that are automatically created. If this is not set explicitly, the default is the filename that was passed to the MobiLink server by the client.	2
s.file_size	INTEGER. This optional IN parameter can be used to limit the size of file that can be uploaded.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	3
s.subdir	VARCHAR(128). Required. This optional INOUT parameter sets the subdirectory location for the files to be transferred. To use the root directory, set this option to null. This option must not include ellipsis (three dots), comma, forward slash (/) or backslash (\). This defaults to remote_key if it is not set explicitly.	Not applicable

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_key	VARCHAR(128). Optional IN parameter to specify a remote key for the file transfer.	Not applicable

Remarks

The MobiLink server executes this event before allowing any download file transfer using the mlfiletransfer utility or MLFileUpload method. It is executed after the user has authenticated using regular authentication. If this script is not defined, the file transfer is allowed.

The MLFileUpload method can only be used by UltraLite clients.

See also

- "Adding and deleting scripts" on page 285
- "-ftr mlsrv12 option" on page 50
- "-ftru mlsrv12 option" on page 50
- "MobiLink File Transfer utility (mlfiletransfer)" [*MobiLink - Client Administration*]
- "MLFileUpload method" [*UltraLite - C and C++ Programming*]
- UltraLite: "Using MobiLink file transfers" [*UltraLite - Database Management and Reference*]
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

authenticate_parameters connection event

Receives values from the remote that can be used to authenticate beyond a user ID and password. The values can also be used to arbitrarily customize each synchronization.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.authentication_status	INTEGER. This is an INOUT parameter.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
a.*N* (one or more)	VARCHAR(128). For example, named parameters could be `a.1 a.2`.	3...

Parameter description

- **authentication_status** The authentication_status parameter is required. It indicates the overall success of the authentication, and can be set to one of the following values:

Returned Value	authentication_status	Description
V <= 1999	1000	Authentication succeeded.
1000 <= V <= 2999	2000	Authentication succeeded, but password expiring soon.
2000 <= V <= 3999	3000	Authentication failed as password has expired.
4000 <= V <= 4999	4000	Authentication failed.
5000 <= V <= 5999	5000	Unable to authenticate because the remote ID is already in use. Try the synchronization again later.
6000 <= V	4000	If the returned value is greater than 5999, MobiLink interprets it as a returned value of 4000 (authentication failed).

- **username** This parameter is the MobiLink user name. VARCHAR(128).
- **remote_ID** The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.

  See "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*].
- **remote a.N** The Nth authentication parameter sent up from the remote client.

Remarks

The number of remote parameters must match the number expected by the authenticate_parameters script or an error results. An error also occurs if parameters are sent from the client and there is no script for this event.

You can send strings (or parameters in the form of strings) from both SQL Anywhere and UltraLite clients. This allows you to have authentication beyond a user ID and password. It also means that you can customize your synchronization based on the value of parameters, and do this in a pre-synchronization phase, during authentication. These parameters may also be referenced from any synchronization script.

The MobiLink server executes this event upon starting each synchronization. It is executed in the same transaction as the authenticate_user event.

You can use this event to replace the built-in MobiLink authentication mechanism with a custom mechanism. You may want to call into the authentication mechanism of your DBMS, or you may want to implement features not present in the MobiLink built-in mechanism.

If the authenticate_user or authenticate_user_hashed scripts are invoked and return an error, this event is not called.

SQL scripts for the authenticate_parameters event must be implemented as stored procedures.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “Authentication parameters” on page 281
- “MobiLink users” [*MobiLink - Client Administration*]
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Custom user authentication” [*MobiLink - Client Administration*]
- “authenticate_user connection event” on page 315
- “authenticate_user_hashed connection event” on page 320
- “begin_synchronization connection event” on page 335
- dbmlsync: “-ap dbmlsync option” [*MobiLink - Client Administration*]
- UltraLite: “Authentication Parameters synchronization parameter” [*UltraLite - Database Management and Reference*] and “Number of Authentication Parameters parameter” [*UltraLite - Database Management and Reference*]

Examples

For UltraLite remote databases, pass the parameters using the num_auth_parms and auth_parms fields in the ul_sync_info struct. num_auth_parms is a count of the number of parameters, from 0 to 255. auth_parms is a pointer to an array of strings. To prevent the strings from being viewed as plain text, the strings are sent using the same obfuscation as passwords. If num_auth_parms is 0, set auth_parms to null. The following is an example of passing parameters in UltraLite:

```
ul_char * Params[ 3 ] = { UL_TEXT( "param1" ),
   UL_TEXT( "param2" ), UL_TEXT( "param3" ) };

...
info.num_auth_parms = 3;
info.auth_parms = Params;
```


For SQL Anywhere remote databases, you pass parameters using the dbmlsync -ap option, in a comma-separated list. For example, the following command line passes three parameters:

```
dbmlsync -ap "param1,param2,param3"
```

In this example, the authenticate_parameters script could be:

```
CALL my_auth_parm (
  {ml s.authentication_status},
  {ml s.remote_id},
  {ml s.username},
  {ml a.1},
  {ml a.2},
  {ml a.3}
)
```

authenticate_user connection event

Implements custom user authentication.

Parameters

In the following table, the description indicates the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.authentication_status	INTEGER. This is an INOUT parameter.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.password	VARCHAR(128). The password for authentication purposes. If the user does not supply a password, this value is null.	3
s.new_password	VARCHAR(128). The new password, if this is being used to reset the user password. If the user does not change their password, this value is null.	4

Default action

Use MobiLink built-in user authentication mechanism.

Remarks

The MobiLink server executes this event upon starting each synchronization. It is executed in a transaction before the begin_synchronization transaction.

You can use this event to replace the built-in MobiLink authentication mechanism with a custom mechanism. You may want to call into the authentication mechanism of your DBMS, or you may want to implement features not present in the MobiLink built-in mechanism, such as password expiry or a minimum password length.

The parameters used in an authenticate_user event are as follows:

- **authentication_status** The authentication_status parameter is required. It indicates the overall success of the authentication, and can be set to one of the following values:

Returned Value	authentication_status	Description
V <= 1999	1000	Authentication succeeded.
2000 <= V <= 2999	2000	Authentication succeeded: password expiring soon.
3000 <= V <= 3999	3000	Authentication failed: password expired.
4000 <= V <= 4999	4000	Authentication failed.
5000 <= V <= 5999	5000	Unable to authenticate because the remote ID is already in use. Try the synchronization again later.

Returned Value	authentica-tion_status	Description
6000 <= V	4000	If the returned value is greater than 5999, MobiLink interprets it as a returned value of 4000.

The value is sent to the client so it can be used to customize authentication behavior at the client.

See:

- "Authentication Value synchronization parameter" [*UltraLite - Database Management and Reference*]
- "sp_hook_dbmlsync_upload_end" [*MobiLink - Client Administration*]

- **username** This optional parameter is the MobiLink user name.

  See "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*].

- **remote_id** The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.

- **password** This optional parameter indicates the password for authentication purposes. If the user does not supply a password, this is null.

- **new_password** This optional parameter indicates a new password. If the user does not change their password, this is null.

SQL scripts for the authenticate_user event must be implemented as stored procedures.

When the two authentication scripts are both defined, and both scripts return different authentication_status codes, the higher value is used.

The authenticate_user script is executed in a transaction along with all authentication scripts. This transaction always commits.

There are predefined scripts that you can use for the authenticate_user event to simplify authentication using LDAP, IMAP and POP3 servers.

See "Authenticating to external servers" [*MobiLink - Client Administration*].

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “MobiLink users” [*MobiLink - Client Administration*]
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Custom user authentication” [*MobiLink - Client Administration*]
- “Authenticating to external servers” [*MobiLink - Client Administration*]
- “authenticate_user_hashed connection event” on page 320
- “authenticate_parameters connection event” on page 312
- “modify_user connection event” on page 407
- “begin_synchronization connection event” on page 335

SQL example

A typical authenticate_user script is a call to a stored procedure. The order of the parameters in the call must match the order above. The following example uses ml_add_connection_script to assign the event to a stored procedure called my_auth.

```
CALL ml_add_connection_script(
'ver1', 'authenticate_user', 'call my_auth ( {ml s.authentication_status},
{ml s.username} )'
)
```

The following SQL Anywhere stored procedure uses only the user name to authenticate—it has no password check. The procedure ensures only that the supplied user name is one of the employee IDs listed in the ULEmployee table.

```
CREATE PROCEDURE my_auth( inout @auth_status int, in @user_name
varchar(128) )
BEGIN
  IF EXISTS
  ( SELECT * FROM ulemployee
    WHERE emp_id = @user_name )
  THEN
    MESSAGE 'OK' type info to client;
    SET @auth_status = 1000;
  ELSE
    MESSAGE 'Not OK' type info to client;
    SET @auth_status = 4000;
  END IF
END
```

Java example

The following call to a MobiLink system procedure registers a Java method called authenticateUser as the script for the authenticate_user event when synchronizing the script version ver1. This syntax is for SQL Anywhere consolidated databases.

```
CALL ml_add_java_connection_script(
   'ver1', 'authenticate_user',
   'ExamplePackage.ExampleClass.authenticateUser'
)
```

The following is the sample Java method authenticateUser. It calls Java methods that check and, if needed, change the user's password.


```
public String authenticateUser(
  ianywhere.ml.script.InOutInteger authStatus,
  String user,
  String pwd,
  String newPwd )
  throws java.sql.sqlException {
  // A real authenticate_user handler would
  // handle more authentication code states.
  _curUser = user;
  if( checkPwd( user, pwd ) ) {
    // Authentication successful.
    if( newPwd != null ) {
      // Password is being changed.
      if( changePwd( user, pwd, newPwd ) ) {
        // Authentication OK and password change OK.
        // Use custom code.
        authStatus.setValue( 1001 );
      } else {
        // Authentication OK but password
        // change failed. Use custom code.
        java.lang.System.err.println( "user: "
         + user + " pwd change failed!" );
        authStatus.setValue( 1002 );
      }
    } else {
      authStatus.setValue( 1000 );
    }
  } else {
    // Authentication failed.
    authStatus.setValue( 4000 );
  }
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called AuthUser as the script for the authenticate_user connection event when synchronizing the script version ver1. This syntax is for SQL Anywhere consolidated databases.

```
CALL ml_add_dnet_connection_script(
   'ver1', 'authenticate_user',
   'TestScripts.Test.AuthUser'
)
```

The following is the sample .NET method AuthUser. It calls .NET methods that check and, if needed, change the user's password.

```
public string AuthUser(
  ref int authStatus,
  string user,
  string pwd,
  string newPwd ) {
  // A real authenticate_user handler would
  // handle more authentication code states.
  _curUser = user;
  if( CheckPwd( user, pwd ) ) {
    // Authentication successful.
    if( newPwd != null ) {
      // Password is being changed.
      if( ChangePwd( user, pwd, newPwd ) ) {
        // Authentication OK and password change OK.
```

```
            // Use custom code.
            authStatus = 1001;
          } else {
            // Authentication OK but password
            // change failed. Use custom code.
            System.Console.WriteLine( "user: "
             + user + " pwd change failed!" );
            authStatus = 1002;
          }
        } else {
          authStatus = 1000 ;
        }
      } else {
        // Authentication failed.
        authStatus = 4000;
      }
      return ( null );
    }
```

For a more detailed example of an authenticate_user script written in C# in .NET, see ".NET synchronization example" on page 600.

authenticate_user_hashed connection event

Implements a custom user authentication mechanism.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.authentication_status	INTEGER. This is an INOUT parameter.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.username	VARCHAR(128). The MobiLink user name.	2
s.hashed_password	BINARY(32). If the user does not supply a password, this value is null.	3
s.hashed_new_password	BINARY(32). If this event is not being used to change the user's password, this value is null.	4

Default action

Use MobiLink built-in user authentication mechanism.

Remarks

This event is identical to authenticate_user except for the passwords, which are in the same hashed form as those stored in the ml_user.hashed_password column. Passing the passwords in hashed form provides increased security.

A one-way hash is used. A one-way hash takes a password and converts it to a byte sequence that is (essentially) unique to each possible password. The one-way hash lets password authentication take place without having to store the actual password in the consolidated database.

Due to incremental improvements in the quality of the hash across MobiLink versions, this script can be called multiple times during an authentication sequence for a user.

When authenticate_user and authenticate_user_hashed are both defined, and both scripts return different authentication_status codes, the higher value is used.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “MobiLink users” [*MobiLink - Client Administration*]
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Custom user authentication” [*MobiLink - Client Administration*]
- “authenticate_user connection event” on page 315
- “authenticate_parameters connection event” on page 312

SQL example

A typical authenticate_user_hashed script is a call to a stored procedure. The order of the parameters in the call must match the order above. The following example calls ml_add_connection_script to assign the event to a stored procedure called my_auth.

```
CALL ml_add_connection_script(
   'ver1', 'authenticate_user_hashed',
   'call my_auth (
```

```
        {ml s.authentication_status},
        {ml s.username},
        {ml s.hashed_password})'
)
```

The following SQL Anywhere stored procedure uses both the user name and password to authenticate. The procedure ensures only that the supplied user name is one of the employee IDs listed in the ULEmployee table. The procedure assumes that the Employee table has a binary(20) column called hashed_pwd.

```
CREATE PROCEDURE my_auth(
  inout @authentication_status integer,
  in @user_name varchar(128),
  in @hpwd binary(32) )
BEGIN
  IF EXISTS
  ( SELECT * FROM ulemployee
    WHERE emp_id = @user_name
      and hashed_pwd = @hpwd )
  THEN
    message 'OK' type info to client;
    RETURN 1000;
  ELSE
    message 'Not OK' type info to client;
    RETURN 4000;
  END IF
END
```

Java example

The following call to a MobiLink system procedure registers a Java method called authUserHashed as the script for the authenticate_user_hashed event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1', 'authenticate_user_hashed',
   'ExamplePackage.ExampleClass.authUserHashed')
```

The following is the sample Java method authUserHashed. It calls Java methods that check and, if needed, change the user's password.

```
public String authUserHashed(
  ianywhere.ml.script.InOutInteger authStatus,
  String user,
  byte pwd[],
  byte newPwd[] )
  throws java.sql.SQLException {
  // A real authenticate_user_hashed handler
  // would handle more auth code states.
  _curUser = user;
  if( checkPwdHashed( user, pwd ) ) {
    // Authorization successful.
    if( newPwd != null ) {
      // Password is being changed.
      if( changePwdHashed( user, pwd, newPwd ) ) {
        // Authorization OK and password change OK.
        // Use custom code.
        authStatus.setValue( 1001 );
      } else {
        // Auth OK but password change failed.
        // Use custom code
        java.lang.System.err.println( "user: " + user
```



```
            + " pwd change failed!" );
          authStatus.setValue( 1002 );
        }
        } else {
          authStatus.setValue( 1000 );
        }
      } else {
        // Authorization failed.
        authStatus.setValue( 4000 );
      }
    return ( null );
  }
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called AuthUserHashed as the script for the authenticate_user_hashed connection event when synchronizing the script version ver1. This syntax is for SQL Anywhere consolidated databases.

```
CALL ml_add_dnet_connection_script(
   'ver1',
   'authenticate_user_hashed',
   'TestScripts.Test.AuthUserHashed'
)
```

The following is the sample .NET method AuthUserHashed.

```
public string AuthUserHashed(
  ref int authStatus,
  string user,
  byte[] pwdHash,
  byte[] newPwdHash ) {
  // A real authenticate_user_hashed handler
  // would handle more auth code states.
  _curUser = user;
  if( CheckPwdHashed( user, pwdHash ) ) {
    // Authorization successful.
    if( newPwdHash != null ) {
      // Password is being changed.
      if( ChangePwdHashed( user, pwdHash, newPwdHash ) ) {
      // Authorization OK and password change OK.
      // Use custom code.
      authStatus = 1001;
    } else {
      // Auth OK but password change failed.
      // Use custom code
      System.Console.WriteLine( "user: " + user
        + " pwd change failed!" );
      authStatus = 1002;
    }
      } else {
        authStatus = 1000;
      }
    } else {
      // Authorization failed.
      authStatus = 4000;
    }
    return ( null );
}
```

begin_connection connection event

Invoked at the time the MobiLink server connects to the consolidated database server.

Parameters

None.

Default action

None.

Remarks

The MobiLink synchronization opens consolidated database connections on demand as synchronization requests come in. When a MobiLink client connects to the MobiLink server, the MobiLink server temporarily allocates one connection with the consolidated database server for all of the database activity for that synchronization. This event may not be called if the MobiLink server is using a connection from the connection pool.

Note
This script is not generally used in Java or .NET, because instead of database variables you would use member variables in this class instance, and prepare the members in the constructor.

See also

- “Adding and deleting scripts” on page 285
- “end_connection connection event” on page 357
- “-cn mlsrv12 option” on page 43
- “-w mlsrv12 option” on page 74

SQL example

The following SQL script works with a SQL Anywhere consolidated database. Two variables are created, one for the last_download timestamp, and one for employee ID.

```
CALL ml_add_connection_script(
    'custdb',
    'begin_connection',
    'create variable @LastDownload timestamp;
    create variable @EmployeeID integer;')
```

begin_connection_autocommit connection event

Invoked at the time MobiLink server connects to the consolidated database server, temporarily allowing you to execute a script when autocommit is on.

Parameters

None.

Default action

Autocommit is off.

Remarks

When the MobiLink server connects to the consolidated database, it turns off autocommit so that it can roll back the upload and download if an error occurs, preserving your data integrity.

However, if you are using an Adaptive Server Enterprise consolidated database, you cannot perform DDL functions such as creating temporary tables unless autocommit is on. If you are using an Adaptive Server Enterprise consolidated database, run your DDL commands in the begin_connection_autocommit event. When the event is finished, autocommit is turned off.

Begin_connection_autocommit scripts must be written so that they are repeatable. This is because if an error or deadlock occurs, the MobiLink server needs to be able to retry the script (since it cannot roll it back).

This event only executes if a script has been defined for the event.

See also

- "Adding and deleting scripts" on page 285

begin_download connection event

Processes any statements just before the MobiLink server commences preparing the download.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_download	TIMESTAMP. The earliest last download time of any synchronized table.	1

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2

Default action

None.

Remarks

The MobiLink server executes this event as the first step in the processing of downloaded information. Download information is processed in a single transaction. The execution of this event is the first action in this transaction.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “end_download connection event” on page 359
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Using last download times in scripts” on page 88

SQL example

The following example calls ml_add_connection_script to assign the event to a stored procedure called SetDownloadParameters.

```
CALL ml_add_connection_script (
   'Lab',
   'begin_download',
   'CALL SetDownloadParameters( {ml s.last_table_download}, {ml
s.username} )' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called beginDownloadConnection as the script for the begin_download connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'example_ver',
   'begin_download',
   'ExamplePackage.ExampleClass.beginDownloadConnection' )
```

The following is the sample Java method beginDownloadConnection. It calls a Java method (prepDeleteTables) that prepares the delete tables using a JDBC connection that was set earlier.


```
public String beginDownloadConnection(
  Timestamp ts,
  String user )
  throws java.sql.SQLException {
  prepDeleteTables ( _syncConn, ts, user );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called BeginDownload as the script for the begin_download connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'begin_download',
  'TestScripts.Test.BeginDownload'
)
```

The following is the sample .NET method BeginDownload. It calls a .NET method (prepDeleteTables) that prepares the delete tables using a database connection that was set earlier.

```
public string BeginDownload(
  DateTime timestamp,
  string user ) {
  prepDeleteTables ( _syncConn, ts, user );
  return ( null );
}
```

begin_download table event

Processes statements related to a specific table just before preparing the download inserts, updates, and deletions.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.table	VARCHAR (128). The table name.	3

Default action

None.

Remarks

The MobiLink server executes this event as the first step in preparing download information for a specific table. The download information is prepared in its own transaction. The execution of this event is the first table-specific action in the download transaction.

You can have one begin_download script for each table in the remote database. The script is only invoked when that table is synchronized.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "end_download table event" on page 361
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]
- "Using last download times in scripts" on page 88

SQL example

The following call to the MobiLink system procedure ml_add_table_script calls the BeginTableDownload procedure. This syntax is for a SQL Anywhere 12 consolidated database.

```
CALL ml_add_table_script(
   'version1',
   'Leads',
   'begin_download',
   'CALL BeginTableDownLoad(
     {ml s.username},
     {ml s.table} )' );
```

The following SQL statements create the BeginTableDownload procedure. It records the download attempt in a table.

```
CREATE PROCEDURE BeginTableDownload(
   MLUser varchar(128),
   TableName varchar(128) )
BEGIN
  INSERT INTO DownloadAttempts ( MLUser, TableName, LastDownload );
END
```


Java example

The following call to a MobiLink system procedure registers a Java method called beginDownloadTable as the script for the begin_download table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_download',
  'ExamplePackage.ExampleClass.beginDownloadTable'
 )
```

The following is the sample Java method beginDownloadTable. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String beginDownloadTable(
  String user,
  String table ) {
  java.lang.System.out.println("Beginning to process download for: " +
table);
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called BeginTableDownload as the script for the begin_download table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'begin_download',
  'TestScripts.Test.BeginTableDownload'
)
```

The following is the sample .NET method BeginTableDownload. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string BeginTableDownload(
  string user,
  string table ) {
  System.Console.WriteLine("Beginning to process download for: " + table);
  return ( null );
}
```

begin_download_deletes table event

Processes statements related to a specific table just before fetching a list of rows to be deleted from the specified table in the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR (128). The MobiLink user name.	2
s.table	VARCHAR (128). The table name.	3

Default action

None.

Remarks

This event is executed immediately before fetching a list of rows to be deleted from the named table in the remote database.

Note
For each download table, the begin_download_deletes, download_delete_cursor, and end_download_deletes events are invoked in sequence. Consider implementing all of the download delete logic for a table in a download_delete_cursor event implemented as a single stored procedure that returns a result set containing all of the rows to be deleted from the remote table. The reduced number of script invocations may result in improved download performance.

You can have one begin_download_deletes script for each table in the remote database.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "begin_download_rows table event" on page 331
- "end_download_rows table event" on page 364
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]
- "Using last download times in scripts" on page 88

begin_download_rows table event

Processes statements related to a specific table just before fetching a list of rows to be inserted or updated in the specified table in the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR (128). The MobiLink user name.	2
s.table	VARCHAR (128). The table name.	3

Default action

None.

Remarks

This event is executed immediately before fetching the rows to be inserted or updated in the named table in the remote database.

Note
For each download table, the begin_download_deletes, download_delete_cursor, and end_download_deletes events are invoked in sequence. Consider implementing all of the download delete logic for a table in a download_delete_cursor event implemented as a single stored procedure that returns a result set containing all of the rows to be deleted from the remote table. The reduced number of script invocations may result in improved download performance.

You can have one begin_download_rows script for each table in the remote database.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "begin_download_deletes table event" on page 329
- "end_download_deletes table event" on page 363
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]
- "Using last download times in scripts" on page 88

begin_publication connection event

Provides useful information about the publication(s) being synchronized. This script may also be used to manage generation numbers for file-based downloads.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.generation_number	INTEGER. This is an INOUT parameter. If your deployment does not use file-based downloads, this parameter can be ignored. The default is 1.	1

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.publication_name	VARCHAR(128). The name of the publication.	3
s.last_publication_upload	TIMESTAMP. The time of the last successful upload of this publication.	4
s.last_publication_download	TIMESTAMP. The last download time for the publication.	5
s.subscription_id	VARCHAR(128). The remote subscription ID.	6

Default action

The default generation number is 1. If no script is defined for this event, the generation number sent to the remote is always 1.

Remarks

This event lets you design synchronization logic based on the publications currently being synchronized. This event is invoked in the same transaction as the begin_synchronization event, and is invoked after the begin_synchronization event. It is invoked once per publication being synchronized.

One potential use for this event is to affect what is downloaded based on the publication used. For example, consider a table that is part of both a priority publication (PriorityPub) and a publication for all tables (AllTablesPub). A script for the begin_publication event could store the publication names in a Java class or a SQL variable or package. Download scripts could then behave differently based on whether the publication being synchronized is PriorityPub or AllTablesPub.

If an UltraLite remote is synchronizing with UL_SYNC_ALL, this event is invoked once with the publication name 'unknown'.

Generation number

The generation_number parameter is specifically for file-based downloads. The output value of the generation number is passed from the begin_publication script to the end_publication script. The meaning of the generation_number depends on whether the current synchronization is being used to create a download file, or whether the current synchronization has an upload.

In file-based downloads, changes to generation numbers are used to force an upload before the download. While generation numbers remain unchanged, remote databases can process many file-based downloads without requiring an upload. The number is stored in the download file. During a synchronization that has an upload, one generation number is output for every subscription to a publication. They are sent to the remote database in the upload acknowledgement, and stored in SYSSYNC.generation_number.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “end_publication connection event” on page 365
- “MobiLink file-based download” on page 247
- “MobiLink generation numbers” on page 253
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Using last download times in scripts” on page 88

SQL example

You may want to record the information for each publication being synchronized. The following example calls ml_add_connection_script to assign the event to a stored procedure called RecordPubSync.

```
CALL ml_add_connection_script(
 'version1',
 'begin_publication',
 '{CALL RecordPubSync(
    {ml s.generation_number},
    {ml s.username},
    {ml s.publication_name},
    {ml s.last_publication_upload},
    {ml s.last_publication_download},
    {ml s.subscription_id} )}' );
```

Java example

The following call to a MobiLink system procedure registers a Java method called beginPublication as the script for the begin_publication connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
 'ver1',
 'begin_publication',
 'ExamplePackage.ExampleClass.beginPublication' )
```

The following is the sample Java method beginPublication. It saves the name of each publication for later use.

```
public String beginPublication(
  ianywhere.ml.script.InOutInteger generation_number,
  String user,
  String pub_name,
  Timestamp last_publication_upload,
  Timestamp last_download ) {
  _publicationNames[ _numPublications++ ] = pub_name;
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called BeginPub as the script for the begin_publication connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'begin_publication',
 'TestScripts.Test.BeginPub'
)
```


The following is the sample .NET method BeginPub. It saves the name of each publication for later use.

```
public string BeginPub(
  ref int generation_number,
  string user,
  string pub_name,
  DateTime last_publication_upload,
  DateTime last_download ) {
  _publicationNames[ _numPublications++ ] = pub_name;
  return ( null );
}
```

begin_synchronization connection event

Processes statements in preparation for the synchronization process.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1

Default action

None.

Remarks

The MobiLink server executes this event after receiving everything from the MobiLink client that is required to begin synchronization.

The begin_synchronization script is useful for maintaining statistics. This is because the end_synchronization script is invoked even if there is an error or conflict, so while the upload transaction is rolled back, things like statistics are maintained.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "end_synchronization connection event" on page 368
- "begin_synchronization table event" on page 337
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

You may want to store the username value in a temporary table or variable if you want to reference that value many times in subsequent scripts.

```
CALL ml_add_connection_script (
  'version1',
  'begin_synchronization',
  'set @EmployeeID = {ml s.username}' );
```

Java example

The following call to a MobiLink system procedure registers a Java method called beginSynchronizationConnection as the script for the begin_synchronization connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
 'ver1',
 'begin_synchronization',
 'ExamplePackage.ExampleClass.beginSynchronizationConnection'
)
```

The following is the sample Java method beginSynchronizationConnection. It saves the name of the synchronizing user for later use.

```
public String beginSynchronizationConnection(
  String user ) {
  _curUser = user;
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called BeginSync as the script for the begin_synchronization connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script( 'ver1',
  'begin_synchronization',
  'TestScripts.Test.BeginSync'
)
```

The following is the sample .NET method BeginSync. It saves the name of the synchronizing user for later use.

```
public string BeginSync(
  string user ) {
```



```
    _curUser = user;
    return ( null );
  }
```

begin_synchronization table event

Processes statements related to a specific table at the beginning of the synchronization.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR (128). The MobiLink user name.	1
s.table	VARCHAR (128). The table name.	2

Default action

None.

Remarks

The MobiLink server executes this event after receiving everything from the MobiLink client that is required to begin synchronization.

You can have one begin_synchronization script for each table in the remote database. The event is only invoked when the table is synchronized.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "end_synchronization table event" on page 370
- "begin_synchronization connection event" on page 335
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

The begin_synchronization table event is used to set up the synchronization of a particular table. The following SQL script registers a script that creates a temporary table for storing rows during synchronization. This syntax is for a SQL Anywhere consolidated database.

```
CALL ml_add_table_script(
 'ver1',
 'sales_order',
 'begin_synchronization',
 'CREATE TABLE #sales_order (
    id          integer NOT NULL default autoincrement,
    cust_id     integer NOT NULL,
    order_date      date NOT NULL,
    fin_code_id char(2) NULL,
    region      char(7) NULL,
    sales_rep       integer NOT NULL,
    PRIMARY KEY (id),
)' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called beginSynchronizationTable as the script for the begin_synchronization table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
 'ver1',
 'table1',
 'begin_synchronization',
 'ExamplePackage.ExampleClass.beginSynchronizationTable' )
```

The following is the sample Java method beginSynchronizationTable. It adds the current table name to a list of table names contained in this instance.

```
public String beginSynchronizationTable(
  String user,
  String table ) {
  _tableList.add( table );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called BeginTableSync as the script for the begin_synchronization table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script (
  'ver1',
  'table1',
```



```
    'begin_synchronization',
    'TestScripts.Test.BeginTableSync' )
```

The following is the sample .NET method BeginTableSync. It adds the current table name to a list of table names contained in this instance.

```
public string BeginTableSync(
  string user,
  string table ) {
  _tableList.Add( table );
  return ( null );
}
```

begin_upload connection event

Processes any statements just before the MobiLink server commences processing the uploaded inserts, updates, and deletes.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR (128). The MobiLink user name.	1

Default action

None.

Remarks

The MobiLink server executes this event as the first step in the processing of uploaded rows. Uploaded rows are processed in a single transaction. The execution of this event is the first action in this transaction.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "end_upload connection event" on page 372
- "begin_upload table event" on page 341
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

The begin_upload connection event is used to perform whatever steps you need performed before uploading rows. The following SQL script creates a temporary table for storing old and new row values for conflict processing of the sales_order table. This example works with a SQL Anywhere consolidated database.

```
CALL ml_add_connection_script(
 'version1',
 'begin_upload',
 'CREATE TABLE #sales_order_conflicts (
    id          integer NOT NULL default autoincrement,
    cust_id     integer NOT NULL,
    order_date      date NOT NULL,
    fin_code_id char(2) NULL,
    region      char(7) NULL,
    sales_rep       integer NOT NULL,
    new_value       char(1) NOT NULL,
    PRIMARY KEY (id) )' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called beginUploadConnection as the script for the begin_upload connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
 'ver1',
 'begin_upload',
 'ExamplePackage.ExampleClass.beginUploadConnection ' )
```

The following is the sample Java method beginUploadConnection. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String beginUploadConnection( String user ) {
  java.lang.System.out.println(
    "Starting upload for user: " + user );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called BeginUpload as the script for the begin_upload connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'begin_upload',
  'TestScripts.Test.BeginUpload'
)
```


The following C# example saves the current user name for use in a later event.

```
public string BeginUpload( string curUser ) {
  user = curUser;
  return ( null );
}
```

begin_upload table event

Processes statements related to a specific table just before the MobiLink server commences processing the uploaded inserts, updates, and deletes.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2

Default action

None.

Remarks

The MobiLink server executes this event as the first step in the processing of uploaded rows. Uploaded rows are processed in a separate transaction. The execution of this event is the first table-specific action in this transaction.

You can have one begin_upload script for each table in the remote database. The script is only invoked when the table is actually synchronized.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “end_upload table event” on page 374
- “begin_upload connection event” on page 339
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

When uploading rows from a remote you may want to place the changes in an intermediate table and manually process changes yourself. You can delete from a global temporary table in this event, in preparation for receiving the new rows.

```
CALL ml_add_table_script(
 'version1',
 'Leads',
 'begin_upload',
 'DELETE FROM T_Leads' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called beginUploadTable as the script for the begin_upload table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_upload',
  'ExamplePackage.ExampleClass.beginUploadTable'
 )
```

The following is the sample Java method beginUploadTable. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String beginUploadTable(
  String user,
  String table ) {
  java.lang.System.out.println("Beginning to process upload for: " + table);
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called BeginTableUpload as the script for the begin_upload table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'begin_upload',
  'TestScripts.Test.BeginTableUpload'
)
```

The following is the sample .NET method BeginTableUpload. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)


```
public string BeginTableUpload(
  string user,
  string table ) {
  System.Console.WriteLine("Beginning to process upload for: " + table);
  return ( null );
}
```

begin_upload_deletes table event

Processes statements related to a specific table just before uploading deleted rows from the specified table in the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2

Default action

None.

Remarks

This event occurs immediately before applying the changes that result from rows deleted from the named remote table.

You can have one begin_upload_deletes script for each table in the remote database. The script is only invoked when the table is actually synchronized.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "end_upload_deletes table event" on page 376
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

The begin_upload_deletes table event is used to perform whatever steps you need performed after uploading inserts and updates for a particular table, but before deletes are uploaded for that table. The following SQL script creates a temporary table for storing deletes temporarily during upload. This syntax is for a SQL Anywhere consolidated database.

```
CALL ml_add_table_script(
 'ver1',
 'sales_order',
 'begin_upload_deletes',
 'CREATE TABLE #sales_order_deletes (
    id          integer NOT NULL default autoincrement,
    cust_id     integer NOT NULL,
    order_date  date NOT NULL,
    fin_code_id char(2) NULL,
    region      char(7) NULL,
    sales_rep   integer NOT NULL,
    PRIMARY KEY (id) )' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called beginUploadDeletes as the script for the begin_upload_deletes table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_upload_deletes',
  'ExamplePackage.ExampleClass.beginUploadDeletes' )
```

The following is the sample Java method beginUploadDeletes. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String beginUploadDeletes(
  String user,
  String table )
  throws java.sql.SQLException {
  java.lang.System.out.println( "Starting upload
    deletes for table: " + table );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called BeginUploadDeletes as the script for the begin_upload_deletes table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
  'ver1',
```



```
  'table1',
  'begin_upload_deletes',
  'TestScripts.Test.BeginUploadDeletes'
)
```

The following is the sample .NET method BeginUploadDeletes. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string BeginUploadDeletes(
  string user,
  string table ) {
  System.Console.WriteLine(
   "Starting upload deletes for table: " + table );
  return ( null );
}
```

begin_upload_rows table event

Processes statements related to a specific table just before uploading inserts and updates from the specified table in the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2

Default action

None.

Remarks

This event occurs immediately before applying the changes that result from inserts and deletes to the remote table named in the second parameter.

You can have one begin_upload_rows script for each table in the remote database. The script is only invoked when the table is actually synchronized.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "end_upload_rows table event" on page 379
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

The begin_upload_rows table event is used to perform whatever steps you need performed before uploading inserts and updates for a particular table. The following script calls a stored procedure that prepares the consolidated database for inserts and updates into the Inventory table:

```
CALL ml_add_table_script(
 'MyCorp 1.0',
 'Inventory',
 'begin_upload_rows',
 'CALL PrepareForUpserts()' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called beginUploadRows as the script for the begin_upload_rows table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_upload_rows',
  'ExamplePackage.ExampleClass.beginUploadRows' )
```

The following is the sample Java method beginUploadRows. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String beginUploadRows(
  String user,
  String table )
  throws java.sql.SQLException {
  java.lang.System.out.println(
   "Starting upload rows for table: " +
   table + " and user: " + user );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called BeginUploadRows as the script for the begin_upload_rows table event when synchronizing the script version ver1 and the table table1.


```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'begin_upload_rows',
  'TestScripts.Test.BeginUploadRows'
)
```

The following is the sample .NET method BeginUploadRows. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string BeginUploadRows(
  string user,
  string table ) {
  System.Console.WriteLine(
   "Starting upload rows for table: " +
   table + " and user: " + user );
  return ( null );
}
```

download_cursor table event

A data script that defines a cursor to select rows to download and insert or update in the given table in the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2

Default action

None.

Remarks

The MobiLink server uses the script to open a read-only cursor to fetch a list of rows to download to the remote database.

You can have one download_cursor script for each table in the remote database.

To optimize performance of the download stage of synchronization to UltraLite clients, when the range of primary key values is outside the current rows on the device, you should order the rows in the download cursor by primary key. Downloads of large reference tables, for example, can benefit from this optimization.

Each download_cursor script must contain a SELECT statement or a call to a procedure that returns a result set. The MobiLink server uses this statement to define a cursor in the consolidated database.

The script must select all columns that correspond to the columns in the corresponding table in the remote database. The columns in the consolidated database can have different names than the corresponding columns in the remote database, but they must be of compatible types.

The columns must be selected in the order that the corresponding columns are defined in the remote database.

To avoid downloading unnecessary rows, consider using timestamp-based downloads. When using timestamp-based downloads, include a line similar to the following in the WHERE clause of your download_cursor script:

```
AND last_modified >= {ml s.last_table_download}
```

This script must be implemented in SQL. For Java or .NET processing of rows, see "Direct row handling" on page 671.

If you are considering using READPAST table hints in download_cursor scripts because you are doing lots of updates that affect download performance, consider using snapshot isolation for downloads instead. The READPAST table hint can cause problems if used in download_cursor scripts. When using timestamp-based downloads, the READPAST hint can cause rows to be missed, and can cause a row to never be downloaded to a remote database. For example:

- A row is added to the consolidated database and committed. The row has a last_modified column with a time of yesterday.
- The same row is updated but not committed.
- A remote database with a last_download time of last week synchronizes.
- A download_cursor script attempts to select the row using READPAST, and skips the row.
- The transaction that updated the row is rolled back. The next last download time for the remote is advanced to today.

From this point on, the row is never downloaded unless it is updated. A possible workaround is to implement a generate_next_last_download_timestamp or modify_next_last_download_timestamp script and set the last download time to be the start time of the oldest open transaction.

See also

- “Data scripts” on page 308
- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “Writing scripts to download rows” on page 290
- “Writing download_cursor scripts” on page 291
- “Partitioning rows among remote databases” on page 92
- “download_delete_cursor table event” on page 349
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Using last download times in scripts” on page 88
- "Using READPAST with MobiLink synchronization" in “FROM clause” [*SQL Anywhere Server - SQL Reference*]

SQL example

The following example comes from an Oracle installation, although the statement is valid against all supported databases. This example downloads all rows that have been changed since the last time the user downloaded data, and that match the user name in the emp_name column.

```
CALL ml_add_table_script(
  'Lab',
  'ULOrder',
  'download_cursor',
  'SELECT order_id,
    cust_id,
    prod_id,
    emp_id,
    disc,
    quant,
    notes,
    status
   FROM ULOrder
   WHERE last_modified >= {ml s.last_table_download}
    AND emp_name = {ml s.username}' )
```

download_delete_cursor table event

A data script that defines a cursor to select rows that are to be deleted in the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown

below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2

Default action

None.

Remarks

The MobiLink server opens a read-only cursor to fetch a list of rows to download, and then delete from the remote database. This script must contain a SELECT statement that returns the primary key values of the rows to be deleted from the table in the remote database.

You can have one download_delete_cursor script for each table in the remote database.

If the download_delete_cursor has nulls for the primary key columns for one or more rows in a table, then MobiLink tells the remote to delete all the rows in the table. See "Deleting all the rows in a table" on page 294.

Note that rows deleted from the consolidated database can not appear in a result set defined by a download_delete_cursor event, and so are not automatically deleted from the remote database. One technique for identifying rows to be deleted from remote databases is to add a column to the consolidated database table identifying a row as inactive.

To avoid downloading unnecessary rows to delete, consider using timestamp-based downloads. Include a line similar to the following in the WHERE clause of your download_delete_cursor script:

```
AND last_modified >= {ml s.last_table_download}
```

This script must be implemented in SQL. For Java or .NET processing of rows, see "Direct row handling" on page 671.

It can be problematic using READPAST table hints in a download_delete_cursor. For details, see the download_cursor event.

See also

- “Data scripts” on page 308
- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “download_cursor table event” on page 347
- “Writing scripts to download rows” on page 290
- “Partitioning rows among remote databases” on page 92
- “Writing download_delete_cursor scripts” on page 293
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Using last download times in scripts” on page 88
- "Using READPAST with MobiLink synchronization" in “FROM clause” [*SQL Anywhere Server - SQL Reference*]

SQL example

This example is taken from the Contact sample and can be found in *Samples\MobiLink\Contact\build_consol.sql*. It deletes from the remote database any customers who have been changed since the last time this user downloaded data (`Customer.last_modified >= {ml s.last_table_download}`), and either

- do not belong to the synchronizing user (`SalesRep.username != {ml s.username}`), or
- are marked as inactive in the consolidated database (`Customer.active = 0`).

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'download_delete_cursor',
 'SELECT cust_id FROM Customer key join SalesRep
  WHERE Customer.last_modified >= {ml s.last_table_download} AND
   ( SalesRep.username != {ml s.username} OR Customer.active = 0 )')
```

download_statistics connection event

Provides access to synchronization statistics for download operations.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark, but you cannot mix names and question marks within a scriptIn SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script.. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name as specified in your SYNCHRONIZATION USER definition.	1
s.warnings	INTEGER. The number of warnings issued.	2
s.errors	INTEGER. The number of errors, including handled errors, that occurred.	3
s.fetched_rows	INTEGER. The number of rows fetched by the download_cursor script.	4
s.deleted_rows	INTEGER. The number of rows fetched by the download_delete_cursor script.	5
s.filtered_rows	INTEGER. The number of rows from the fetched_rows parameter actually sent to the remote. This reflects download filtering of uploaded values.	6
s.bytes	INTEGER. The number of bytes sent to the remote as the download.	7

Default action

None.

Remarks

The download_statistics event allows you to gather, for any user, statistics on downloads. The download_statistics connection script is called just before the commit at the end of the download transaction.

Note
Depending on the command line, not all warnings are logged. The warnings count passed to this script is the number of warnings that would be logged when no warnings are disabled, which may be more than the number of warnings logged.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “download_statistics table event” on page 354
- “upload_statistics connection event” on page 509
- “upload_statistics table event” on page 514
- “synchronization_statistics connection event” on page 424
- “synchronization_statistics table event” on page 427
- “time_statistics connection event” on page 430
- “time_statistics table event” on page 433
- “MobiLink Monitor” on page 154
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following example inserts synchronization statistics into a table called download_audit.

```
CALL ml_add_connection_script(
 'ver1',
 'download_statistics',
 'INSERT INTO download_audit(
   user_name,
   warnings,
   errors,
   fetched_rows,
   deleted_rows,
   filtered_rows,
   bytes )
  VALUES (
   {ml s.username},
   {ml s.warnings},
   {ml s.errors},
   {ml s.fetched_rows},
   {ml s.deleted_rows},
   {ml s.filtered_rows},
   {ml s.bytes})')
```

Once vital statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following call to a MobiLink system procedure registers a Java method called downloadStatisticsConnection as the script for the download_statistics event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
  'ver1',
  'download_statistics',
  'ExamplePackage.ExampleClass.downloadStatisticsConnection' )
```

The following is the sample Java method downloadStatisticsConnection. It prints the number of fetched rows to the MobiLink message log. (Note that printing the number of fetched rows to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String downloadStatisticsConnection(
  String user,
```

```
  int warnings,
  int errors,
  int fetchedRows,
  int deletedRows,
  int filteredRows,
  int bytes ) {
  java.lang.System.out.println(
   "download connection stats fetchedRows: "
   + fetchedRows );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called DownloadStats as the script for the download_statistics connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'download_statistics',
  'TestScripts.Test.DownloadStats'
)
```

The following is the sample .NET method DownloadStats. It prints the number of fetched rows to the MobiLink message log. (Note that printing the number of fetched rows to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string DownloadStats(
  string user,
  int warnings,
  int errors,
  int deletedRows,
  int fetchedRows,
  int downloadRows,
  int filteredRows,
  int bytes ) {
  System.Console.WriteLine(
   "download connection stats fetchedRows: "
   + fetchedRows );
  return ( null );
}
```

download_statistics table event

Provides access to synchronization statistics for download operations by table.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use

parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name as specified in your SYNCHRONIZATION USER definition.	1
s.table	VARCHAR(128). The table name.	2
s.warnings	INTEGER. The number of warnings issued.	3
s.errors	INTEGER. The number of errors, including handled errors, that occurred.	4
s.fetched_rows	INTEGER. The number of rows fetched by the download_cursor script.	5
s.deleted_rows	INTEGER. The number of rows fetched by the download_delete_cursor script.	6
s.filtered_rows	INTEGER. The number of rows filtered from the fetched_rows. This reflects download filtering of uploaded values.	7
s.bytes	INTEGER. The number of bytes sent to the remote as the download.	8

Default action

None.

Remarks

The download_statistics event allows you to gather, for any user and table, statistics on downloads as they apply to that table. The download_statistics table script is called just before the commit at the end of the download transaction.

Note
Depending on the command line, not all warnings are logged. The warnings count passed to this script is the number of warnings that would be logged when no warnings are disabled, which may be more than the number of warnings logged.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “download_statistics connection event” on page 351
- “upload_statistics connection event” on page 509
- “upload_statistics table event” on page 514
- “synchronization_statistics connection event” on page 424
- “synchronization_statistics table event” on page 427
- “time_statistics connection event” on page 430
- “time_statistics table event” on page 433
- “MobiLink Monitor” on page 154
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following example inserts synchronization statistics into a table called download_audit. Once vital statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'download_statistics',
 'INSERT INTO download_audit (
   user_name,
   table, warnings,
   errors,
   fetched_rows,
   deleted_rows,
   filtered_rows,
   bytes)
  VALUES (
   {ml s.username},
   {ml s.table},
   {ml s.warnings},
   {ml s.errors},
   {ml s.fetched_rows},
   {ml s.deleted_rows},
   {ml s.filtered_rows},
   {ml s.bytes})')
```

Java example

The following call to a MobiLink system procedure registers a Java method called downloadStatisticsTable as the script for the download_statistics table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'download_statistics',
  'ExamplePackage.ExampleClass.downloadStatisticsTable' )
```

The following is the sample Java method downloadStatisticsTable. It prints some statistics for this table to the MobiLink message log. (Note that printing statistics for a table to the MobiLink message log might be useful at development time but would slow down a production server.)


```
public String downloadStatisticsTable(
  String user,
  String table,
  int warnings,
  int errors,
  int fetchedRows,
  int deletedRows,
  int filteredRows,
  int bytes ) {
  java.lang.System.out.println( "download table stats "
   + "table: " + table + "bytes: " + bytes );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called DownloadTableStats as the script for the download_statistics table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
 'ver1',
 'table1',
 'download_statistics',
 'TestScripts.Test.DownloadTableStats'
)
```

The following is the sample .NET method DownloadTableStats. It prints some statistics for this table to the MobiLink message log. (Note that printing statistics for a table to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string DownloadTableStats(
  string user,
  string table,
  int warnings,
  int errors,
  int fetchedRows,
  int deletedRows,
  int filteredRows,
  int bytes ) {
  System.Console.WriteLine( "download table stats "
   + "table: " + table + "bytes: " + bytes );
  return ( null );
}
```

end_connection connection event

Processes any statements just before the MobiLink server closes a connection with the consolidated database server, either in preparation to shut down or when a connection is removed from the connection pool.

Parameters

None.

Default action

None.

Remarks

You can use the end_connection script to perform an action of your choice just before closing a connection between the MobiLink server and the consolidated database server.

This script is normally used to complete any actions started by the begin_connection script and free any resources acquired by it.

See also

- "begin_connection connection event" on page 324
- "Adding and deleting scripts" on page 285

SQL example

The following SQL script drops a temporary table that was created by the begin_connection script. This syntax is for a SQL Anywhere consolidated database. Strictly speaking, this table doesn't need to be dropped explicitly, since SQL Anywhere does this automatically when the connection is destroyed. Whether a temporary table needs to be dropped explicitly depends on your consolidated database type.

```
CALL ml_add_connection_script(
 'version 1.0',
 'end_connection',
 'DROP TABLE #sync_info' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called endConnection as the script for the end_connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1',
   'end_connection',
   'ExamplePackage.ExampleClass.endConnection' )
```

The following is the sample Java method endConnection. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String endConnection() {
  java.lang.System.out.println( "Ending connection." );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndConnection as the script for the end_connection connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'end_connection',
  'TestScripts.Test.EndConnection'
)
```


The following is the sample .NET method EndConnection. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string EndConnection() {
  System.Console.WriteLine( "Ending connection." );
  return ( null );
}
```

end_download connection event

Processes any statements just after the MobiLink server concludes preparation of the download data.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_download	TIMESTAMP. The earliest last download time of any synchronized table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2

Default action

None.

Remarks

The MobiLink server executes this script after all download rows have been fetched from the consolidated database. The execution of this script is the last non-statistical action in the download.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “begin_download connection event” on page 325
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Using last download times in scripts” on page 88

SQL example

The following example shows one possible use of an end_download connection script. This script deletes rows from a temporary table used to help generate the download.

```
CALL ml_add_connection_script(
 'ver1',
 'end_download',
 'DELETE FROM TempDownloadTable where user = {ml s.username}')
```

Java example

The following call to a MobiLink system procedure registers a Java method called endDownloadConnection as the script for the end_download connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_download',
  'ExamplePackage.ExampleClass.endDownloadConnection' )
```

The following is the sample Java method endDownloadConnection. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String endDownloadConnection(
  Timestamp ts,
  String user )
{
  java.lang.System.out.println( "Ending download for user: " + user );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndDownload as the script for the end_download connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'end_download',
 'TestScripts.Test.EndDownload' )
```

The following is the sample .NET method EndDownload. It prints a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string EndDownload(
  DateTime timestamp,
  string user ) {
```



```
    System.Console.WriteLine( "Ending download for user: " + user );
    return ( null );
}
```

end_download table event

Processes statements related to a specific table just after the MobiLink server concludes preparing the download rows.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.table	VARCHAR(128). The table name.	3

Default action

None.

Remarks

The MobiLink server executes this script after all rows have been downloaded and confirmation of receipt has been received. The download information is prepared in a separate transaction. The execution of this script is the last table-specific, non-statistical action in this transaction.

You can have one end_download script for each table in the remote database.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "begin_download table event" on page 327
- "end_download connection event" on page 359
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]
- "Using last download times in scripts" on page 88

SQL example

The end_download table event is used to perform whatever steps you need performed after downloading a particular table. The following SQL Anywhere SQL script drops a temporary table created by a prepare_for_download script to hold download rows for the sales_summary table.

```
CALL ml_add_table_script(
 'MyCorp 1.0',
 'sales_summary',
 'end_download',
 'DROP TABLE #sales_summary_download' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called endDownloadTable as the script for the end_download table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script (
  'ver1',
  'table1',
  'end_download',
  'ExamplePackage.ExampleClass.endDownloadTable' )
```

The following is the sample Java method endDownloadTable. It resets the current table member variable.

```
public String endDownloadTable(
  Timestamp ts,
  String user,
  String table ) {
  _curTable = null;
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndTableDownload as the script for the end_download table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'end_download',
  'TestScripts.Test.EndTableDownload'
)
```

The following is the sample .NET method EndTableDownload. It resets the current table member variable.

```
public string EndTableDownload
  DateTime timestamp,
```



```
    string user,
    string table ) {
    _curTable = null;
    return ( null );
}
```

end_download_deletes table event

Processes statements related to a specific table just after preparing a list of rows to be deleted from the specified table in the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.table	VARCHAR(128). The table name.	3

Default action

None.

Remarks

This script is executed immediately after preparing a list of rows to be deleted from the named table in the remote database.

Note
For each download table, the begin_download_deletes, download_delete_cursor, and end_download_deletes events are invoked in sequence. Consider implementing all of the download delete logic for a table in a download_delete_cursor event implemented as a single stored procedure that returns a result set containing all of the rows to be deleted from the remote table. The reduced number of script invocations may result in improved download performance.

You can have one end_download_deletes script for each table in the remote database.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "begin_download_deletes table event" on page 329
- "end_download connection event" on page 359
- "begin_download_rows table event" on page 331
- "end_download_rows table event" on page 364
- "download_delete_cursor table event" on page 349
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]
- "Using last download times in scripts" on page 88

end_download_rows table event

Processes statements related to a specific table just after preparing a list of rows to be inserted or updated in the specified table in the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.table	VARCHAR(128). The table name.	3

Default action

None.

Remarks

This script is executed immediately after preparing the stream of rows to be inserted or updated in the named table in the remote database.

Note
For each download table, the begin_download_deletes, download_delete_cursor, and end_download_deletes events are invoked in sequence. Consider implementing all of the download delete logic for a table in a download_delete_cursor event implemented as a single stored procedure that returns a result set containing all of the rows to be deleted from the remote table. The reduced number of script invocations may result in improved download performance.

You can have one end_download_rows script for each table in the remote database.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “begin_download_rows table event” on page 331
- “end_download connection event” on page 359
- “end_download_deletes table event” on page 363
- “begin_download_deletes table event” on page 329
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Using last download times in scripts” on page 88

end_publication connection event

Provides useful information about the publication(s) being synchronized.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.generation_number	INTEGER. If your deployment does not use file-based downloads, this parameter can be ignored. The default value is 1.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.publication_name	VARCHAR(128)	3
s.last_publication_upload	TIMESTAMP. Last successful upload time of this publication.	4
s.last_publication_download	TIMESTAMP. The last download time of this publication.	5
s.subscription_id	VARCHAR(128). The remote subscription ID.	6

Default action

None.

Remarks

This event lets you design synchronization logic based on the publications currently being synchronized. This event is invoked in the same transaction as the end_synchronization event, and is invoked before the end_synchronization event. It is invoked once per publication being synchronized.

If the current synchronization successfully applied an upload, the last_upload parameter contains the time this latest upload was applied. The last_publication_download is the same value that was passed to the download scripts as the last download time.

If an UltraLite remote is synchronizing with UL_SYNC_ALL, this event is invoked once with the name 'unknown'.

Generation number

The generation_number parameter is specifically for file-based downloads. In file-based downloads, changes to generation numbers are used to force an upload before the download when the file is applied at the remote. The number is stored in the download file.

The output value of the generation number is passed from the begin_publication script to the end_publication script. The meaning of the generation_number depends on whether the current synchronization is being used to create a download file, or whether the current synchronization has an upload.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "begin_publication connection event" on page 332
- "MobiLink file-based download" on page 247
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]
- "Using last download times in scripts" on page 88

SQL example

You may want to record the information for each publication being synchronized. The following example calls ml_add_connection_script to assign the event to a stored procedure called RecordPubEndSync.

```
CALL ml_add_connection_script(
   'version1',
   'end_publication',
   'CALL RecordPubEndSync(
    {ml s.generation_number},
    {ml s.username},
    {ml s.publication_name},
    {ml s.last_publication_upload},
    {ml s.last_publication_download} )' );
```

Java example

The following call to a MobiLink system procedure registers a Java method called endPublication as the script for the end_publication connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1',
   'end_publication',
   'ExamplePackage.ExampleClass.endPublication' )
```

The following is the sample Java method endPublication. It outputs a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String endPublication(
  int generation_number,
  String user,
  String pub_name,
  Timestamp last_publication_upload,
  Timestamp last_publication_download ) {
```

```
    java.lang.System.out.println(
     "Finished synchronizing publication " + pub_name );
    return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndPub as the script for the end_publication connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'end_publication',
  'TestScripts.Test.EndPub'
)
```

The following is the sample .NET method endPub. It outputs a message to the MobiLink message log. (Note that printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string EndPub(
  int generation_number,
  string user,
  string pub_name,
  DateTime last_publication_upload,
  DateTime last_publication_download ) {
  System.Console.Write(
   "Finished synchronizing publication " + pub_name );
  return ( null );
}
```

end_synchronization connection event

Processes statements at the end of the synchronization process.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.synchronization_ok	INTEGER. This value is 1 for a successful synchronization and 0 for an unsuccessful synchronization.	2

Default action

None.

Remarks

The MobiLink server executes this script after synchronization is complete.

This script is executed within a separate transaction after the download transaction. If no download acknowledgement is expected, the remote may finish its synchronization and disconnect before the end_synchronization script begins or completes.

The end_synchronization script is useful for maintaining statistics. This is because if the begin_synchronization script is called, the end_synchronization script is invoked even if there is an error in any previous transaction, so while the upload transaction is rolled back, statistics are maintained.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “begin_synchronization connection event” on page 335
- “begin_synchronization table event” on page 337
- “end_synchronization table event” on page 370
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following SQL script calls a system procedure that records the end time of the synchronization attempt along with its success or failure status. This syntax is for SQL Anywhere consolidated databases.

```
CALL ml_add_connection_script(
 'ver1',
 'end_synchronization',
 'CALL RecordEndOfSyncAttempt(
   {ml s.username},
   {ml s.synchronization_ok} )' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called endSynchronizationConnection as the script for the end_synchronization event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_synchronization',
  'ExamplePackage.ExampleClass.endSynchronizationConnection'
)
```

The following is the sample Java method endSynchronizationConnection. It uses a JDBC connection to execute an update. This syntax is for SQL Anywhere consolidated databases.

```
public String endSynchronizationConnection(
  String user )
  throws java.sql.SQLException {
  execUpdate( _syncConn,
   "UPDATE sync_count set count = count + 1 where user_id = '"
   + user + "' " );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndSync as the script for the end_synchronization connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'end_synchronization',
  'TestScripts.Test.EndSync'
)
```

The following is the sample .NET method EndSync. It updates the table sync_count. This syntax is for SQL Anywhere consolidated databases.

```
public string EndSync(
  string user ) {
  return(
   "UPDATE sync_count set count = count + 1 where user_id = '"
   + user + "' ");
  return ( null );
}
```

end_synchronization table event

Processes statements at the end of the synchronization process.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.synchronization_ok	INTEGER. This value is 1 for a successful synchronization and 0 for an unsuccessful synchronization.	3

Default action

None.

Remarks

The MobiLink server executes this script after an application has synchronized and is about to disconnect from the MobiLink server, and before the connection level script of the same name.

You can have one end_synchronization script for each table in the remote database.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “begin_synchronization table event” on page 337
- “end_synchronization connection event” on page 368
- “end_synchronization table event” on page 370
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following SQL Anywhere SQL script drops a temporary table created by the begin_synchronization script.

```
CALL ml_add_table_script(
 'ver1',
 'sales_order',
```

```
  'end_synchronization',
  'DROP TABLE #sales_order' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called endSynchronizationTable as the script for the end_synchronization table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
   'ver1',
   'table1',
   'end_synchronization',
   'ExamplePackage.ExampleClass.endSynchronizationTable' )
```

The following is the sample Java method endSynchronizationTable. It returns a SQL statement that drops a temporary table created by the begin_synchronization script.

```
public String endSynchronizationTable(
  String user,
  String table ) {
  return( "DROP TABLE #sales_order" );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndTableSync as the script for the end_synchronization table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
 'ver1',
 'table1',
 'end_synchronization',
 'TestScripts.Test.EndTableSync'
)
```

The following is the sample .NET method EndTableSync. It returns a SQL to statement that drops a temporary table created by the begin_synchronization script.

```
public string EndTableSync(
  string user,
  string table ) {
  return( "DROP TABLE #sales_order" );
}
```

end_upload connection event

Processes any statements just after the MobiLink server concludes processing uploaded inserts, updates, and deletes.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1

Default action

None.

Remarks

The MobiLink server executes this script as the last step in the processing of uploaded information. Upload information is processed in a single transaction. The execution of this script is the last action in this transaction before statistical scripts.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “begin_upload connection event” on page 339
- “end_upload table event” on page 374
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following SQL Anywhere SQL script calls the EndUpload stored procedure.

```
CALL ml_add_connection_script(
 'ver1',
 'sales_order',
 'end_upload',
 'CALL EndUpload({ml s.username});' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called endUploadConnection as the script for the end_upload connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1',
```

```
    'end_upload',
    'ExamplePackage.ExampleClass.endUploadConnection' )
```

The following is the sample Java method endUploadConnection. It calls a method to perform operations on the database.

```
public String endUploadConnection( String user ) {
  // Clean up new and old tables.
  Iterator    two_iter = _tables_with_ops.iterator();
  while( two_iter.hasNext() ) {
    TableInfo cur_table = (TableInfo)two_iter.next();
    dumpTableOps( _sync_conn, cur_table );
  }
  _tables_with_ops.clear();
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndUpload as the script for the end_upload connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'end_upload',
  'TestScripts.Test.EndUpload'
)
```

The following is the sample .NET method EndUpload. It returns a SQL statement that calls the EndUpload stored procedure.

```
public string EndUpload( string user ) {
  return ( "CALL EndUpload({ml s.username});" );
}
```

end_upload table event

Processes statements related to a specific table just after the MobiLink server concludes processing of uploaded inserts, updates, and deletions.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2

Default action

None.

Remarks

The MobiLink server executes this script as the last step in the processing of uploaded information. Upload information is processed in a separate transaction. The execution of this script is the last table-specific action in this transaction.

You can have one end_upload script for each table in the remote database.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “begin_upload table event” on page 341
- “end_upload connection event” on page 372
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following call to a MobiLink system procedure assigns the end_upload event to a stored procedure called ULCustomerIDPool_maintain.

```
CALL ml_add_table_script(
   'custdb',
   'ULCustomerIDPool',
   'end_upload',
   '{ CALL ULCustomerIDPool_maintain( {ml s.username} ) }' )
```

The following SQL statements create the ULCustomerIDPool_maintain stored procedure. This procedure inserts new primary keys, to replace the keys used by the rows just uploaded, into a primary key pool that gets downloaded to the remote database later in the same synchronization.

```
CREATE PROCEDURE ULCustomerIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
    DECLARE pool_count INTEGER;

    -- Determine how many ids to add to the pool
    SELECT COUNT(*) INTO pool_count
```

```
        FROM ULCustomerIDPool WHERE pool_emp_id = syncuser_id;

    -- Top up the pool with new ids
    WHILE pool_count < 20 LOOP
        INSERT INTO ULCustomerIDPool ( pool_emp_id ) VALUES ( syncuser_id );
        SET pool_count = pool_count + 1;
    END LOOP;
END
```

Java example

The following call to a MobiLink system procedure registers a Java method called endUploadTable as the script for the end_upload table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
   'ver1',
   'table1',
   'end_upload',
   'ExamplePackage.ExampleClass.endUploadTable' )
```

The following is the sample Java method endUploadTable. It generates a delete for a table with a name related to the passed-in table name. This syntax is for SQL Anywhere consolidated databases.

```
public String endUploadTable(
  String user,
  String table ) {
  return( "DELETE from '" + table + "_temp'" );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndUpload as the script for the end_upload table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
   'ver1',
   'table1',
   'end_upload',
   'TestScripts.Test.EndUpload'
)
```

The following .NET example moves rows inserted into a temporary table into the table passed into the script.

```
public string EndUpload( string user, string table ) {
  DBCommand stmt = curConn.CreateCommand();
  // Move the uploaded rows to the destination table.
  stmt.CommandText = "INSERT INTO "
    + table
    + " SELECT * FROM dnet_ul_temp";
  stmt.ExecuteNonQuery();
  stmt.Close();
  return ( null );
}
```

end_upload_deletes table event

Processes statements related to a specific table just after applying deletes uploaded from the specified table in the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2

Default action

None.

Remarks

This script is run immediately after applying the changes that result from rows deleted in the given remote table.

You can have one end_upload_deletes script for each table in the remote database.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "begin_upload_deletes table event" on page 343
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

You can use this event to process rows deleted during the upload on an intermediate table. You can compare the rows in the base table with rows in the intermediate table and decide what to do with the deleted row.

The following call to a MobiLink system procedure assigns the EndUploadDeletesLeads stored procedure to the end_upload_deletes event.

```
CALL ml_add_table_script(
  'version1',
  'Leads',
  'end_upload_deletes',
  'call EndUploadDeletesLeads()');
```

The following SQL statement creates the EndUploadDeletes stored procedure.

```
CREATE PROCEDURE EndUploadDeletesLeads ( )
Begin
  FOR names AS curs CURSOR FOR
  SELECT LeadID
    FROM Leads
    WHERE LeadID NOT IN (SELECT LeadID FROM T_Leads);
  DO
   CALL decide_what_to_do( LeadID )
  END FOR;
end
```

Java example

The following call to a MobiLink system procedure registers a Java method called endUploadDeletes as the script for the end_upload_deletes table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
   'ver1',
   'table1',
   'end_upload_deletes',
   'ExamplePackage.ExampleClass.endUploadDeletes' )
```

The following is the sample Java method a endUploadDeletes. It calls a Java method that manipulates the database.

```
public String endUploadDeletes(
  String user,
  String table )
  throws java.sql.SQLException {
  processUploadedDeletes( _syncConn, table );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndUploadDeletes as the script for the end_upload_deletes table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
   'ver1',
   'table1',
   'end_upload_deletes',
   'TestScripts.Test.EndUploadDeletes'
)
```

The following is the sample .NET method a EndUploadDeletes. It calls a .NET method that manipulates the database.

```
public string EndUploadDeletes(
  string user,
  string table ) {
```



```
    processUploadedDeletes( _syncConn, table );
    return ( null );
}
```

end_upload_rows table event

Processes statements related to a specific table just after applying uploaded inserts and updates from the specified table in the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2

Default action

None.

Remarks

This script is run immediately after applying the changes that result from modifications to the given remote table.

You can have one end_upload_rows script for each table in the remote database.

See also

- "Adding and deleting scripts" on page 285
- "begin_upload_rows table event" on page 345
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL Example

The following call to a MobiLink system procedure registers a SQL method called EndUploadRows as the script for the end_upload_rows table event when synchronizing the script version ver1.

```
CALL ml_add_table_script(
  'version1',
  'table1',
  'end_upload_rows',
  'CALL EndUploadRows(
      { ml s.username },
      { ml s.table } )' )
```

The following is the sample SQL method EndUploadRows. It calls a SQL method that manipulates the database.

```
CREATE PROCEDURE EndUploadRows (
 IN user VARCHAR(128)
 IN table VARCHAR{128} )
BEGIN
   CALL decide_what_to_do(table);
END;
```

Java example

The following call to a MobiLink system procedure registers a Java method called endUploadRows as the script for the end_upload_rows table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
   'ver1',
   'table1',
   'end_upload_rows',
   'ExamplePackage.ExampleClass.endUploadRows' )
```

The following is the sample Java method endUploadRows. It calls a Java method that manipulates the database.

```
public String endUploadRows(
  String user,
  String table )
  throws java.sql.SQLException {
  processUploadedRows( _syncConn, table );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndUploadRows as the script for the end_upload_rows table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
  'ver1',
```



```
  'table1',
  'end_upload_rows',
  'TestScripts.Test.EndUploadRows'
)
```

The following is the sample .NET method endUploadRows. It calls a .NET method that manipulates the database.

```
public string EndUploadRows(
  string user,
  string table ) {
  processUploadedRows( _syncConn, table );
  return ( null );
}
```

generate_next_last_download_timestamp event

The script is used to invoke a user-defined algorithm to generate the next_last_download_timestamp.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.next_last_download	TIMESTAMP. This is an INOUT parameter. The MobiLink server initializes this parameter with the last_download_timestamp, a timestamp used to generate a download stream in the current synchronization.	1
s.username	VARCHAR(128). The MobiLink user name.	2

Remarks

This script is invoked in the prepare_for_download transaction, right before the prepare_for_download script is called.

Use this event with caution, especially with consolidated databases that support a snapshot isolation level, such as, SQL Anywhere, Oracle, Microsoft SQL Server, and IBM DB2 LUW 9.7. The MobiLink server always uses the snapshot isolation level for download with Oracle. By default, it also uses the snapshot isolation level for download with SQL Anywhere and Microsoft SQL Server, when the snapshot isolation level is enabled on the database.

For robust timestamp-based synchronization, the output of next_last_download must be the earlier of:

1. the current timestamp

2. the starting timestamp of the earliest open transaction updating (for example, inserting, updating or deleting) any table or view used to construct the download.

This script can also be specified as an ignored script using the --{ml_ignore} clause. When this script is defined as an ignored script, the MobiLink server does not call this script and does not use MobiLink internal logic to generate the next last download timestamp. Instead, the MobiLink server sends back to the client the last download timestamp that was sent by the client in the current synchronization. You can use this technique for synchronizations that always download all the rows from the consolidated database for all the synchronization tables. However, for time-based synchronization, you should define this script as a real script using the appropriate business logic to generate the next last download timestamp. Alternatively, don't define any script for this event and the MobiLink server uses its internal logic to generate the next last download timestamp.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Using last download times in scripts” on page 88
- “How download timestamps are generated and used” on page 88
- “modify_next_last_download_timestamp connection event” on page 405
- “modify_last_download_timestamp connection event” on page 402

SQL example

The generate_next_last_download_timestamp script can be used in the MobiLink server to generate UTC time-based downloads. Here are the steps to set up a UTC time based download for Oracle:

To set up a UTC time based download for Oracle

1. Assume you have a sync table called **my_table** that is defined as follows:

```
CREATE TABLE my_table (  pk              INT PRIMARY KEY NOT NULL,
                         c1              VARCHAR(100) ,
                         last_modified   TIMESTAMP DEFAULT
SYS_EXTRACT_UTC( SYSTIMESTAMP )
)
```


2. Create a stored procedure called **GenerateNextDownloadTimestamp** to get the starting time of the earliest open transaction in UTC in the Oracle database:

```
CREATE PROCEDURE GenerateNextDownloadTimestamp ( p_ts IN OUT TIMESTAMP )
AS
BEGIN
     SELECT SYS_EXTRACT_UTC( NVL( MIN( TO_TIMESTAMP( START_TIME, 'mm/dd/rr
hh24:mi:ss' ) ),
                                        SYSTIMESTAMP ) )
          INTO p_ts FROM GV$TRANSACTION;
END;
```

3. Call the ml_add_connect_script to install the script:

```
call ml_add_connection_script(
  'my_script_version',
  'generate_next_last_download_timestamp',
  '{ call GenerateNextDownloadTimestamp( {ml s.next_last_download} ) }' )
```

Note
The MobiLink server logon ID must have a select permission on GV$TRANSACTION.

handle_DownloadData connection event

A non-SQL data script used by direct row handling to create a set of rows to download.

Parameters

None.

Default action

None.

Remarks

The handle_DownloadData event allows you to determine what operations to download to MobiLink clients using direct row handling.

Direct row handling is used to synchronize to data sources other than MobiLink supported consolidated databases. See “Direct row handling” on page 671.

To create the direct download, you can use the DownloadData and DownloadTableData classes in the MobiLink server API for Java or .NET.

For Java, the DBConnectionContext getDownloadData method returns a DownloadData instance for the current synchronization. DownloadData encapsulates all download operations to send to a remote client. You can use the DownloadData getDownloadTables and getDownloadTableByName methods to obtain a DownloadTableData instance. DownloadTableData encapsulates download operations for a particular table. You can use the getUpsertPreparedStatement method to obtain prepared statements for insert and update operations. You can use the DownloadTableData getDeletePreparedStatement method to obtain prepared statements for delete operations.

For .NET, the DBConnectionContext GetDownloadData method returns a DownloadData instance for the current synchronization. DownloadData encapsulates all download operations to send to a remote client. You can use the DownloadData GetDownloadTables and GetDownloadTableByName methods to obtain a DownloadTableData instance. DownloadTableData encapsulates download operations for a particular table. You can use the GetUpsertCommand method to obtain commands for insert and update operations. You can use the DownloadTableData getDeleteCommand method to obtain commands for delete operations.

For Java, see "DBConnectionContext interface" on page 536. For .NET, see "DBConnectionContext interface" on page 614.

You can create the download in handle_DownloadData or another synchronization event. MobiLink provides this flexibility so that you can set the download when data is uploaded or when particular events occur. If you want to create the direct download in an event other than handle_DownloadData, you must create a handle_DownloadData script whose method does nothing. Except in upload-only synchronization, the MobiLink server requires that at a minimum, a handle_DownloadData script be defined to enable direct row handling of downloads.

If you create the direct download in an event other than handle_DownloadData, the event must not be before the begin_synchronization event and cannot be after the end_download event.

Note
This event cannot be implemented as SQL.

See also

- "Data scripts" on page 308
- "Direct row handling" on page 671
- "Handling direct downloads" on page 681
- Java: "DownloadData interface" on page 542
- Java: "DownloadTableData interface" on page 544
- .NET: "DownloadData interface" on page 635
- .NET: "DownloadTableData interface" on page 637
- "handle_UploadData connection event" on page 394
- "Required scripts" on page 284
- "Adding and deleting scripts" on page 285

Java example

The following call to a MobiLink system procedure registers a Java method called handleDownload for the handle_DownloadData connection event when synchronizing the script version ver1. You run this system procedure against your MobiLink consolidated database.

```
CALL ml_add_java_connection_script(
 'ver1',
 'handle_DownloadData',
 'MyPackage.MobiLinkOrders.handleDownload' )
```

See "ml_add_java_connection_script system procedure" on page 698.

The following example shows you how to use the handleDownload method to create a download.

The following code sets up a class level DBConnectionContext instance in the constructor for a class called MobiLinkOrders.

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;
import java.lang.System;

public class  MobiLinkOrders{

  DBConnectionContext _cc;

  public MobiLinkOrders( DBConnectionContext cc ) {
    _cc = cc;
  }
```

In your HandleDownload method, you use the DBConnectionContext getDownloadData method to return a DownloadData instance for the current synchronization. The DownloadData getDownloadTableByName method returns a DownloadTableData instance for the remoteOrders table. The DownloadTableData getUpsertPreparedStatement method returns a java.sql.PreparedStatement. To add an operation to the download, you set all column values and call the executeUpdate method.

The following is the handleDownload method of the MobiLinkOrders class. It adds two rows to the download for a table called remoteOrders.

```
// Method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
  // Get DownloadData instance for current synchronization.
  DownloadData downloadData = _cc.getDownloadData();

  // Get a DownloadTableData instance for the remoteOrders table.
  DownloadTableData td = downloadData.getDownloadTableByName("remoteOrders");

  // Get a java.sql.PreparedStatement for upsert (update/insert) operations.
  PreparedStatement upsertPS = td.getUpsertPreparedStatement();

  // Set values for one row.
  upsertPS.setInt( 1, 2300 );
  upsertPS.setInt( 2, 100 );

  // Add the values to the download.
  int updateResult = upsertPS.executeUpdate();

  // Set values for another row.
  upsertPS.setInt( 1, 2301 );
  upsertPS.setInt( 2, 50 );
  updateResult = upsertPS.executeUpdate();
  // ...
  upsertPS.close();

}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called HandleDownload as the script for the handle_DownloadData connection event when synchronizing the script version ver1. This syntax is for SQL Anywhere consolidated databases.

```
CALL ml_add_dnet_connection_script(
   'ver1', 'handle_DownloadData',
```

```
   'TestScripts.MobiLinkOrders.HandleDownload'
)
```

The following is the sample .NET method HandleDownload:

```
using System;
using System.Data;
using System.IO;
using iAnywhere.MobiLink.Script;
using iAnywhere.MobiLink;

namespace MyScripts
{
    /// <summary>
    /// Tests that scripts are called correctly for most sync events.
    /// </summary>
    public class MobiLinkOrders
    {
 private DBConnectionContext _cc;

 public MobiLinkOrders( DBConnectionContext cc )
 {
     _cc = cc;
 }

 ~MobiLinkOrders()
 {
 }

 public void handleDownload()
 {
     // Get DownloadData instance for current synchronization.
     DownloadData my_dd = _cc.GetDownloadData();

     // Get a DownloadTableData instance for the remoteOrders table.
     DownloadTableData td = my_dd.GetDownloadTableByName("remoteOrders");

     // Get an IDbCommand for upsert (update/insert) operations.
     IDbCommand upsert_stmt = td.GetUpsertCommand();

     IDataParameterCollection parameters = upsert_stmt.Parameters;

     // Set values for one row.
     parameters[ 0 ] = 2300;
     parameters[ 1 ] = 100;

     // Add the values to the download.
     int update_result = upsert_stmt.ExecuteNonQuery();

     // Set values for another row.
     parameters[ 0 ] = 2301;
     parameters[ 1 ] = 50;
     update_result = upsert_stmt.ExecuteNonQuery();

     // ...
 }

    }
}
```


handle_error connection event

Executed whenever the MobiLink server encounters a SQL error while invoking a data script.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.action_code	INTEGER. This is an INOUT parameter. Set this value to tell MobiLink server how to respond to the error.	1
s.error_code	INTEGER. The native RDBMS error code.	2
s.error_message	TEXT. The native RDBMS error message.	3
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	4
s.table	VARCHAR(128). The table whose script had the error. If the script is not a table script, the table name is null.	5

Default action

The MobiLink server selects a default action. You can modify the action in the script, and return a value instructing MobiLink how to proceed. The action_code parameter takes one of the following values:

- **1000** Skip the current row and continue processing.

- **3000** Rollback the current transaction and cancel the current synchronization. This is the default action code, and is used when no handle_error script is defined or this script causes an error.

- **4000** Rollback the current transaction, cancel the synchronization, and shut down the MobiLink server.

Remarks

The MobiLink server sends in the current action code. Initially, this is set to 3000 for each set of errors caused by a single SQL operation. Usually, there is only one error per SQL operation, but there may be more. If uploading rows in batches, this handle_error script is called once per error in the batch. During the same synchronization the action code passed into the first error is 3000. Subsequent calls are passed in the action code returned by the previous call. MobiLink uses the highest numerical value returned from multiple calls.

For more information about uploading rows in batches, see "-s mlsrv12 option" on page 64.

You can modify the action code in the script, and return a value instructing MobiLink how to proceed. The action code tells the MobiLink server what to do next. Before it calls this script, the MobiLink server sets the action code to a default value, which depends upon the severity of the error. Your script may modify this value. Your script must return or set an action code.

The error_code and message allow you to identify the nature of the error.

The MobiLink server executes this script if an ODBC error occurs while MobiLink is processing an insert, update, or delete script during the upload transaction or is fetching download rows. If an ODBC error occurs at another time, the MobiLink server calls the report_error or report_ODBC_error script and aborts the synchronization.

If the error happened while manipulating a particular table, the table name is supplied. Otherwise, this value is null. The table name is the name of a table in the client application. This name may or may not have a direct counterpart in the consolidated database, depending upon how your remote table names map to consolidated tables.

SQL scripts for the handle_error event must be implemented as stored procedures.

You can return a value from the handle_error script one of the following ways:

- Pass the action_code parameter to an OUTPUT parameter of a procedure:

```
CALL my_handle_error( {ml s.action_code}, {ml s.error_code}, {ml
s.error_message}, {ml s.username}, {ml s.table} )
```

- Set the action_code via a procedure or function return value:

```
{ml s.action_code} = CALL my_handle_error( {ml s.error_code}, {ml
s.error_message}, {ml s.username}, {ml s.table} )
```

  Most RDBMSs use the RETURN statement to set the return value from a procedure or function.

The CustDB sample application contains error handlers for various database-management systems.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “report_error connection event” on page 416
- “report_odbc_error connection event” on page 419
- “handle_odbc_error connection event” on page 391
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Data scripts” on page 308

SQL example

The following example works with a SQL Anywhere consolidated database. It allows your application to ignore redundant inserts.

The following call to a MobiLink system procedure assigns the ULHandleError stored procedure to the handle_error event.

```
CALL ml_add_connection_script(
    'ver1',
    'handle_error',
    'CALL ULHandleError(
      {ml s.action_code},
      {ml s.error_code},
      {ml s.error_message},
      {ml s.username},
      {ml s.table} )' )
```

The following SQL statement creates the ULHandleError stored procedure.

```
CREATE PROCEDURE ULHandleError(
   INOUT action integer,
   IN error_code integer,
   IN error_message varchar(1000),
   IN user_name varchar(128),
   IN table_name varchar(128) )
BEGIN
  -- -196 is SQLE_INDEX_NOT_UNIQUE
  -- -194 is SQLE_INVALID_FOREIGN_KEY
  IF error_code = -196 or error_code = -194 then
      -- ignore the error and keep going
      SET action = 1000;
  ELSE
      -- abort the synchronization
      SET action = 3000;
  END IF;
END
```

Java example

The following call to a MobiLink system procedure registers a Java method called handleError as the script for the handle_error connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
    'ver1',
    'handle_error',
    'ExamplePackage.ExampleClass.handleError' )
```

The following is the sample Java method handleError. It processes an error based on the data that is passed in. It also determines the resulting error code.

```
public String handleError(
  ianywhere.ml.script.InOutInteger actionCode,
  int errorCode,
  String errorMessage,
  String user,
  String table ) {
  int newAC;
  if( user == null ) {
    newAC = handleNonSyncError( errorCode,
    errorMessage ); }
  else if( table == null ) {
    newAC = handleConnectionError( errorCode,
    errorMessage, user ); }
  else {
    newAC = handleTableError( errorCode,
    errorMessage, user, table );
  }
  // Keep the most serious action code.
  if( actionCode.getValue() < newAC ) {
    actionCode.setValue( newAC );
  }
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called HandleError as the script for the handle_error connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'handle_error',
  'TestScripts.Test.HandleError' )
```

The following is the sample .NET method HandleError.

```
public string HandleError() (
  ref int actionCode,
  int errorCode,
  string errorMessage,
  string user,
  string table ) {
  int new_ac;
  if( user == null ) {
    new_ac = HandleNonSyncError( errorCode,
    errorMessage ); }
  else if( table == null ) {
    new_ac = HandleConnectionError( errorCode,
    errorMessage, user ); }
  else {
    new_ac = HandleTableError( errorCode,
    errorMessage, user, table );
  }
  // Keep the most serious action code.
  if( actionCode < new_ac ) {
    actionCode = new_ac;
  }
}
```


handle_odbc_error connection event

Executed whenever the MobiLink server encounters an ODBC error while invoking a data script.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.action_code	INTEGER. This is an INOUT parameter. Set this value to tell the MobiLink server how to respond to the error.	1
s.odbc_state	VARCHAR(5). The ODBC SQLSTATE.	2
s.error_message	TEXT. The ODBC error message	3
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	4
s.table	VARCHAR(128)	5

Default action

The MobiLink server selects a default action. You can modify the action in the script, and return a value instructing MobiLink how to proceed. The action_code parameter takes one of the following values:

- **1000** Skip the current row and continue processing.
- **3000** Rollback the current transaction and cancel the current synchronization. This is the default action code, and is used when no handle_error script is defined or this script causes an error.

- **4000** Rollback the current transaction, cancel the synchronization, and shut down the MobiLink server.

Remarks

The MobiLink server executes this script whenever it encounters an error flagged by the ODBC Driver Manager if the error occurs while MobiLink is processing an insert, update, or delete script during the upload transaction or is fetching download rows. If an ODBC error occurs at another time, the MobiLink server calls the report_error or report_ODBC_error script and aborts the synchronization.

The error codes allow you to identify the nature of the error.

The action code tells the MobiLink server what to do next. Before it calls this script, the MobiLink server sets the action code to a default value, which depends upon the severity of the error. Your script may modify this value. Your script must return or set an action code.

The handle_odbc_error script is called after the handle_error and report_error scripts, and before the report_odbc_error script.

When only one, but not both, error-handling script is defined, the return value from that script decides error behavior. When both error-handling scripts are defined, the MobiLink server uses the numerically highest action code. If both handle_error and handle_ODBC_error are defined, MobiLink uses the action code with the highest numerical value returned from all calls.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "handle_error connection event" on page 387
- "report_error connection event" on page 416
- "report_odbc_error connection event" on page 419
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

The following example works with a SQL Anywhere consolidated database. It allows your application to ignore ODBC integrity constraint violations.

The following call to a MobiLink system procedure assigns the HandleODBCError stored procedure to the handle_odbc_error event.

```
CALL ml_add_connection_script(
 'ver1',
 'handle_odbc_error',
 'CALL HandleODBCError(
   {ml s.action_code},
   {ml s.ODBC_state},
   {ml s.error_message},
   {ml s.username},
   {ml s.table} )' )
```

The following SQL statement creates the HandleODBCError stored procedure.

```
CREATE PROCEDURE HandleODBCError(
   INOUT action integer,
   IN odbc_state varchar(5),
```



```
   IN error_message varchar(1000),
   IN user_name varchar(128),
   IN table_name varchar(128) )
  BEGIN
   IF odbc_state = '23000' then
      -- Ignore the error and keep going.
      SET action = 1000;
   ELSE
      -- Abort the synchronization.
      SET action = 3000;
   END IF;
END
```

Java example

The following call to a MobiLink system procedure registers a Java method called handleODBCError as the script for the handle_odbc_error event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
 'ver1',
 'handle_odbc_error',
 'ExamplePackage.ExampleClass.handleODBCError'
)
```

The following is the sample Java method handleODBCError. It processes an error based on the data that is passed in. It also determines the resulting error code.

```
public String handleODBCError(
  ianywhere.ml.script.InOutInteger actionCode,
  String ODBCState,
  String errorMessage,
  String user,
  String table ) {
  int newAC;
     newAC = handleTableError( ODBCState,
     errorMessage, user, table );
   // Keep the most serious action code.
  if( actionCode.getValue() < newAC ) {
    actionCode.setValue( newAC );
  }
  return( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called HandleODBCError as the script for the handle_odbc_event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'handle_odbc_error',
 'TestScripts.Test.HandleODBCError' )
```

The following is the sample .NET method HandleODBCError.

```
public string HandleODBCError (
  ref int actionCode,
  string ODBCState,
  string errorMessage,
  string user,
  string table ) {
```

```
    int new_ac;
       new_ac = HandleTableError( ODBCState,
       errorMessage, user, table );
     // Keep the most serious action code.
    if( actionCode < new_ac ) {
      actionCode = new_ac;
    }
    return( null );
  }
```

handle_UploadData connection event

A non-SQL data script used by direct row handling to process uploaded rows.

Parameters

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
UploadData	A .NET or Java class encapsulating table operations uploaded by a MobiLink client. This class is defined in the MobiLink server API for Java and .NET.	1

Default action

None.

Remarks

The handle_UploadData event allows you to process the upload for MobiLink direct row handling. This event fires once for each upload transaction in a synchronization, unless you are using transaction-level uploads, in which case it fires for each transaction.

See "Direct row handling" on page 671.

This event takes a single UploadData parameter. Your Java or .NET method can use the UploadData getUploadedTables or getUploadedTableByName methods to obtain UploadedTableData instances. UploadedTableData allows you to access insert, update, and delete operations uploaded by a MobiLink client in the current synchronization.

For more information about the UploadData and UploadedTableData classes, see "Handling direct uploads" on page 675.

If you want to read column name metadata, you should specify the SendColumnNames MobiLink client extended option or property. As of version 12, SendColumnNames is on by default. Optionally, you can establish column names using the ml_add_column system procedure. Otherwise you can refer to columns by index, as defined at the remote database.

See "SendColumnNames (scn) extended option" [*MobiLink - Client Administration*] and "Send Column Names synchronization parameter" [*UltraLite - Database Management and Reference*].

To get the uploaded pre-image columns for an update, use the SetOldRowValues and SetNewRowValues methods. See "Handling direct upload conflicts" on page 676.

Note
This event cannot be implemented as SQL.

See also

- "Data scripts" on page 308
- "Direct row handling" on page 671
- "Handling direct uploads" on page 675
- Java: "UploadData interface" on page 580
- Java: "UploadedTableData interface" on page 582
- .NET: "UploadData interface" on page 660
- .NET: "UploadedTableData interface" on page 662
- dbmlsync: "SendColumnNames (scn) extended option" [*MobiLink - Client Administration*]
- UltraLite: "Send Column Names synchronization parameter" [*UltraLite - Database Management and Reference*]
- "handle_DownloadData connection event" on page 383
- "Required scripts" on page 284
- "Adding and deleting scripts" on page 285

Java examples

The following call to a MobiLink system procedure registers a Java method called handleUpload for the handle_UploadData connection event when synchronizing the script version ver1. You run this system procedure against your MobiLink consolidated database.

```
CALL ml_add_java_connection_script(
  'ver1',
  'handle_UploadData',
  'MyPackage.MyClass.handleUpload' )
```

For more information about ml_add_java_connection_script, see "ml_add_java_connection_script system procedure" on page 698.

The following Java method processes the upload for the remoteOrders table. The UploadData.getUploadedTableByName method returns an UploadedTableData instance for the remoteOrders table. The UploadedTableData getInserts method returns a java.sql.ResultSet instance representing new rows.

```
import ianywhere.ml.script.*;
import java.sql.*;
import java.io.*;
// ...

public void handleUpload( UploadData ut )
  throws SQLException, IOException {
  // Get an UploadedTableData instance representing the
  // remoteOrders table.
  UploadedTableData remoteOrdersTable =
ut.getUploadedTableByName("remoteOrders");
  // Get inserts uploaded by the MobiLink client.
  java.sql.ResultSet results = remoteOrdersTable.getInserts();
```

```
    while( results.next() ) {
      // You can reference column names here because SendColumnNames is on.
      // Get the primary key.
      int pk = results.getInt("pk");

      // Get the uploaded num_ordered value.
      int numOrdered = results.getInt("num_ordered");

      // The current insert row is now ready to be uploaded to wherever
      // you want it to go (a file, a web service, and so on).

    }

    results.close();

  }
```

The following example outputs insert, update and delete operations uploaded by a MobiLink remote database. The UploadData getUploadedTables method returns UploadedTableData instances representing all tables uploaded by a remote. The order of the tables in this array is the order in which they where uploaded by the remote. The UploadedTableData getInserts, getUpdates, and getDeletes methods return standard JDBC result sets. You can use the println method or output data to a text file or another location.

```
import ianywhere.ml.script.*;
import java.sql.*;
import java.io.*;
// ...

public void handleUpload( UploadData ud )
  throws SQLException, IOException {
  UploadedTableData tables[] = ud.getUploadedTables();
  for( int i = 0; i < tables.length; i++ ) {
    UploadedTableData currentTable = tables[i];
    println( "table " + java.lang.Integer.toString( i ) +
      " name: " + currentTable.getName() );
    // Print out insert result set.
    println( "Inserts" );
    printRSInfo( currentTable.getInserts() );
    // print out update result set
    println( "Updates" );
    printUpdateRSInfo( currentTable.getUpdates() );
    // Print out delete result set.
    println( "Deletes" );
    printRSInfo( currentTable.getDeletes() );
  }
}
```

The printRSInfo method prints out an insert, update, or delete result set and accepts a single java.sql.ResultSet object. Detailed column information, including column labels, is provided by the ResultSetMetaData object returned by the ResultSet getMetaData method. Column labels are available only if the client has the SendColumnNames option turned on. The printRow method prints out each row in a result set.

```
public void printRSInfo( ResultSet results )
  throws SQLException, IOException {

  // Obtain the result set metadata.
  ResultSetMetaData metaData = results.getMetaData();
  String columnHeading = "";
```



```
    // Print out column headings.
    for( int c = 1; c <= metaData.getColumnCount(); c++ ) {
      columnHeading += metaData.getColumnLabel(c);
      if( c < metaData.getColumnCount() ) {
        columnHeading += ", ";
      }
    }

    println( columnHeading );
    while( results.next() ) {

      // Print out each row.
      printRow( results, metaData.getColumnCount() );
    }

    // Close the java.sql.ResultSet.
    results.close();
  }
```

The printRow method, shown below, uses the ResultSet getString method to obtain each column value.

```
  public void printRow( ResultSet results, int colCount )
    throws SQLException, IOException {
    String row = "( ";

    for( int c = 1; c <= colCount; c++ ) {
      // Get a column value.
      String currentColumn = results.getString( c );

      // Check for null values.
      if( currentColumn == null ) {
        currentColumn = "<NULL>";
      }

      // Add the column value to the row string.
      row += cur_col;
      if( c < colCount ) {
        row += ", ";
      }
    }

    row += " )";

    // Print out the row.
    println( row );
  }
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called HandleUpload for the handle_UploadData connection event when synchronizing the script version ver1. You run this system procedure against your MobiLink consolidated database.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'handle_UploadData',
  'TestScripts.Test.HandleUpload' )
```

The following .NET method processes the upload for the remoteOrders table. This example makes use of the SetOldRowValues and SetNewRowValues methods to access both the pre-image and post-image of each update.

```
using System;
using System.Data;
using System.IO;
using iAnywhere.MobiLink.Script;
using iAnywhere.MobiLink;

namespace MyScripts
{
    public class MyUpload
    {
 public MyUpload( DBConnectionContext cc )
 {
 }

 ~MyUpload()
 {
 }

 public void handleUpload( UploadData ut )
 {
     int i;
     UploadedTableData[] tables = ut.GetUploadedTables();

     for( i=0; i<tables.Length; i+=1 ) {
  UploadedTableData cur_table = tables[i];
  Console.Write( "table " + i + " name: " + cur_table.GetName() );

  // Print out insert result set.
  Console.Write( "Inserts" );
  printRSInfo( cur_table.GetInserts() );

  // print out update result set
  Console.Write( "Updates" );
  printUpdateRSInfo( cur_table.GetUpdates() );

  // Print out delete result set.
  Console.Write( "Deletes" );
  printRSInfo( cur_table.GetDeletes() );
     }
 }

 public void printRSInfo( IDataReader dr )
 {
     // Obtain the result set metadata.
     DataTable dt = dr.GetSchemaTable();
     DataColumnCollection cc = dt.Columns;
     DataColumn dc;
     String columnHeading = "";

     // Print out column headings.
     for( int c=0; c < cc.Count; c = c + 1 ) {
  dc = cc[ c ];
  columnHeading += dc.ColumnName;
  if( c < cc.Count - 1 ) {
      columnHeading += ", ";
  }
     }
     Console.Write( columnHeading );

     while( dr.Read() ) {
  // Print out each row.
  printRow( dr, cc.Count );
     }
```



```
    // Close the java.sql.ResultSet.
    dr.Close();
}

public void printUpdateRSInfo( UpdateDataReader utr )
{
    // Obtain the result set metadata.
    DataTable dt = utr.GetSchemaTable();
    DataColumnCollection cc = dt.Columns;
    DataColumn dc;
    String columnHeading = "TYPE, ";

    // Print out column headings.
    for( int c = 0; c < cc.Count; c = c + 1 ) {
 dc = cc[ c ];
 columnHeading += dc.ColumnName;
 if( c < cc.Count - 1 ) {
     columnHeading += ", ";
 }
    }
    Console.Write( columnHeading );

    while( utr.Read() ) {
 // Print out the new values for the row.
 utr.SetNewRowValues();
 Console.Write( "NEW:" );
 printRow( utr, cc.Count );

 // Print out the old values for the row.
 utr.SetOldRowValues();
 Console.Write( "OLD:" );
 printRow( utr, cc.Count );
    }

    // Close the java.sql.ResultSet.
    utr.Close();
}

public void printRow( IDataReader dr, int col_count )
{
    String row = "( ";
    int c;

    for( c = 0; c < col_count; c = c + 1 ) {
 // Get a column value.
 String cur_col = dr.GetString( c );

 // Check for null values.
 if( cur_col == null ) {
     cur_col = "<NULL>";
 }

 // Add the column value to the row string.
 row += cur_col;
 if( c < col_count ) {
     row += ", ";
 }
    }

    row += " )";

    // Print out the row.
    Console.Write( row );
}
```

```
        }
    }
```

modify_error_message connection event

The script can be used to customize the message text (error, warning, and information) that is sent to remote databases.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.error_message	VARBINARY(1024). This is an IN-OUT parameter, respresenting the error message.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.error_code	INT. The MobiLink error code associated with the error_message.	3

Default action

None.

Remarks

This script gives you the ability to change the error_message into something the remote user and/or application can understand better than the original message.

SQL scripts for the modify_error_message event must be implemented as stored procedures.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

The following example downloads everything from one day ago, regardless of whether the databases were synchronized since then.

The following SQL statement creates the ModifyLastErrorMessage stored procedure:

```
CREATE PROCEDURE ModifyLastErrorMessage(
  inout error_message VARBINARY(1024),
  in username VARCHAR(128),
  in error_code INT )
BEGIN
  SELECT dateadd(day, -1, last_download_time )
  INTO last_download_time
END
```

The following call to a MobiLink system procedure assigns ModifyLastErrorMessage to the modify_error_message connection event for the script version modify_ts_test:

```
CALL ml_add_connection_script(
  'modify_ts_test',
  'modify_error_message',
  'CALL ModifyLastErrorMessage (
    {ml s.error_message},
    {ml s.username},
    {ml s.error_code} )' );
```

Java example

The following call to a MobiLink system procedure registers a Java method called modifyLastErrorMessage as the script for the modify_error_message connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
  'ver1',
  'modify_error_message',
  'ExamplePackage.ExampleClass.modifyLastErrorMessage' )
```

The following is the sample Java method modifyLastErrorMessage. It prints the current error message and error code.

```
public String modifyLastErrorMessage(
  ianywhere.ml.script.InOutString lastErrorMessage,
  String userName,
  int errorCode ) {
  java.lang.System.out.println( "error message: " +
   lastErrorMessage );
  java.lang.System.out.println( "error code: " +
   String.valueOf(errorCode) );
  return( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called ModifyLastErrorMessage as the script for the modify_error_message connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'modify_error_message',
  'TestScripts.Test.ModifyLastErrorMessage' )
```

The following is a sample .NET method ModifyLastErrorMessage. It prints the current error code and error message.

```
public string ModifyLastErrorMessage (
  ref string errorMessage,
  string userName,
  string errorCode  ) {
  System.Console.WriteLine( "error message: " + errorMessage );
  System.Console.WriteLine( "error code: " +  errorCode );
  return ( null );
}
```

modify_last_download_timestamp connection event

The script can be used to modify the last download time for the current synchronization.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_download	TIMESTAMP. The earliest last download time for any synchronized table. This is an INOUT parameter.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.username	VARCHAR(128). The MobiLink user name.	2

Default action

None.

Remarks

Use this script when you want to modify the last_download timestamp for the current synchronization. If this script is defined, the MobiLink server uses the modified last_download timestamp as the last_download timestamp passed to the download scripts. A typical use of this script is to recover from losing data on the remote; you can reset the last_download timestamp to an early time such as 1900-01-01 00:00 so that the next synchronization downloads all the data. Also, when updates to consolidated tables can be time-stamped with times earlier than the time of the actual update, for example via DBMS replication, this script lets you adjust the last download time to avoid missing these updates on download.

SQL scripts for the modify_last_download_timestamp event must be implemented as stored procedures.

This script is executed just before the prepare_for_download script, in the same transaction.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Using last download times in scripts” on page 88
- “How download timestamps are generated and used” on page 88
- “modify_next_last_download_timestamp connection event” on page 405
- “generate_next_last_download_timestamp event” on page 381

SQL example

The following SQL statement creates a stored procedure. The following syntax is for Oracle consolidated databases:

```
CREATE PROCEDURE ModifyDownloadTimestamp(
        download_timestamp  OUT DATE,
        user_name   IN VARCHAR )
        AS
        BEGIN
         -- N is the maximum replication latency in the consolidated cluster
        download_timestamp := download_timestamp - N;
        END;
```

The following syntax is for SQL Anywhere, Adaptive Server Enterprise, and SQL Server consolidated databases:

```
CREATE PROCEDURE ModifyDownloadTimestamp
        @download_timestamp DATETIME OUTPUT,
        @user_name    VARCHAR( 128 )
```

```
AS
BEGIN
 -- N is the maximum replication latency in consolidated cluster
SELECT @download_timestamp = @download_timestamp - N
END
```

The following call to a MobiLink system procedure assigns the ModifyDownloadTimestamp stored procedure to the modify_last_download_timestamp event. The following syntax is for a SQL Anywhere consolidated database:

```
CALL ml_add_connection_script(
  'my_version',
  'modify_last_download_timestamp',
  '{CALL ModifyDownloadTimestamp(
   {ml s.last_download},
   {ml s.username} ) }' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called modifyLastDownloadTimestamp as the script for the modify_last_download_timestamp connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
  'ver1',
  'modify_last_download_timestamp',
  'ExamplePackage.ExampleClass.modifyLastDownloadTimestamp' )
```

The following is the sample Java method modifyLastDownloadTimestamp. It prints the current and new timestamp and modifies the timestamp that is passed in.

```
public String modifyLastDownloadTimestamp(
     Timestamp lastDownloadTime,
     String userName ) {
   java.lang.System.out.println( "old date: " +
    lastDownloadTime.toString() );
   lastDownloadTime.setDate(
    lastDownloadTime.getDate() -1 );
   java.lang.System.out.println( "new date: " +
    lastDownloadTime.toString() );
   return( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called ModifyLastDownloadTimestamp as the script for the modify_last_download_timestamp connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'modify_last_download_timestamp',
  'TestScripts.Test.ModifyLastDownloadTimestamp' )
```

The following is the sample .NET method ModifyLastDownloadTimestamp.

```
public string ModifyLastDownloadTimestamp(
  ref DateTime lastDownloadTime,
  string userName ) {
  System.Console.WriteLine( "old date: " +
```



```
    last_download_time.ToString() );
  last_download_time = DateTime::Now;
  System.Console.WriteLine( "new date: " +
    last_download_time.ToString() );
  return( null );
}
```

modify_next_last_download_timestamp connection event

The script can be used to modify the last download time for the next synchronization.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.next _last_download	TIMESTAMP. This is an INOUT parameter. The MobiLink server generates this value immediately after the upload is committed.	1
s.last_download	TIMESTAMP. This is the last download time for the current synchronization.	2
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	3

Default action

None.

Remarks

This script allows you to change the next_last_download timestamp, which effectively changes the last_download timestamp for the next synchronization. This allows you to reset the next synchronization without affecting the current synchronization. During normal synchronization, the next_last_download is later than than, but also sometimes equal to, the last_download time.

SQL scripts for the modify_next_last_download_timestamp event must be implemented as stored procedures. The MobiLink server passes in the next_last_download timestamp as the first parameter to the stored procedure, and replaces the timestamp by the first value passed out by the stored procedure.

This script is executed in the download transaction, after downloading user tables, but the output value of your stored procedure should correspond to the beginning of the download transaction so that rows changed during the download transaction are downloaded on the next synchronization.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]
- "Using last download times in scripts" on page 88
- "How download timestamps are generated and used" on page 88
- "modify_last_download_timestamp connection event" on page 402
- "generate_next_last_download_timestamp event" on page 381

SQL example

The following example shows one application of this script. Create a stored procedure. The following syntax is for a SQL Anywhere consolidated database:

```
CREATE PROCEDURE ModifyNextDownloadTimestamp(
  inout next_last_download TIMESTAMP ,
  in last_download TIMESTAMP ,
  in user_name VARCHAR(128) )
  BEGIN
    SELECT dateadd(hour, -1, next_last_download )
    INTO next_last_download
END
```

Install the script into your SQL Anywhere consolidated database:

```
CALL ml_add_connection_script(
  'modify_ts_test',
  'modify_next_last_download_timestamp',
  'CALL  ModifyNextDownloadTimestamp (
   {ml s.next_last_download},
   {ml s.last_download},
   {ml s.username} )' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called modifyNextDownloadTimestamp as the script for the modify_next_last_download_timestamp connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1',
```



```
    'modify_next_last_download_timestamp',
    'ExamplePackage.ExampleClass.modifyNextDownloadTimestamp' )
```

The following is the sample Java method modifyNextDownloadTimestamp. It sets the download timestamp back an hour.

```
public String modifyNextDownloadTimestamp(
  Timestamp NextLastDownload,
  Timestamp lastDownload,
  String userName ) {
  NextLastDownload.setHours(
  NextLastDownload.getHours() -1 );
  return( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called ModifyNextDownloadTimestamp as the script for the modify_next_last_download_timestamp connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
   'ver1',
   'modify_next_last_download_timestamp',
   'TestScripts.Test.ModifyNextDownloadTimestamp' )
```

The following is the sample .NET method ModifyNextDownloadTimestamp. It sets the download timestamp back an hour.

```
public string ModifyNextDownloadTimestamp (
  ref DateTime next_last_download,
  DateTime last_download,
  string user_name ) {
  next_last_download = next_last_download.AddHours( -1 );
  return ( null );
}
```

modify_user connection event

Modify the MobiLink user name.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name. This is an INOUT parameter.	1

Default action

None.

Remarks

This script is invoked at the end of the authentication transaction.

The MobiLink server provides the user name as a parameter when it calls scripts; the user name is sent by the MobiLink client. Sometimes you may want to have an alternate user name. This script allows you to modify the user name used in calling MobiLink scripts.

The username parameter must be long enough to hold the user name.

SQL scripts for the modify_user event must be implemented as stored procedures.

Note
A more flexible approach to mapping the MobiLink user name is to use user-defined named parameters. See "User-defined named parameters" on page 280.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "authenticate_user connection event" on page 315
- "authenticate_user_hashed connection event" on page 320
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

The following example maps a remote database user name to the ID of the user using the device, by using a mapping table called user_device. This technique can be used when the same person has multiple remotes (such as a PDA and a laptop) requiring the same synchronization logic (based on the user's name or id).

The following call to a MobiLink system procedure assigns the ModifyUser stored procedure to the modify_user event. This syntax is for a SQL Anywhere consolidated database.

```
CALL ml_add_connection_script(
 'ver1',
```



```
 'modify_user',
 'call ModifyUser( {ml s.username} )' )
```

The following SQL statement creates the ModifyUser stored procedure.

```
CREATE PROCEDURE ModifyUser( INOUT u_name varchar(128) )
BEGIN
 SELECT user_name
    INTO u_name
    FROM user_device
    WHERE device_name = u_name;
END
```

Java example

The following call to a MobiLink system procedure registers a Java method called modifyUser as the script for the modify_user connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1',
   'modify_user',
   'ExamplePackage.ExampleClass.modifyUser' )
```

The following is the sample Java method modifyUser. It gets the user ID from the database and then uses it to set the user name.

```
public String modifyUser(
  InOutString ioUserName )
  throws SQLException {
  Statement uidSelect = curConn.createStatement();
  ResultSet uidResult = uidSelect.executeQuery(
   "SELECT rep_id FROM SalesRep WHERE name = '" +
   ioUserName.getValue() + "' " );
  uidResult.next();
  ioUserName.setValue(
   java.lang.Integer.toString(uidResult.getInt( 1 ));
   uidResult.close();
   uidSelect.close();
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called ModUser as the script for the modify_user connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'modify_user',
  'TestScripts.Test.ModUser'
)
```

The following is the sample .NET method ModUser.

```
public string ModUser(
  string user ) {
  return ( "SELECT rep_id FROM SalesRep WHERE name = '" + user + "' " );
}
```

nonblocking_download_ack connection event

When you use download acknowledgement, this script provides a place to record the information that a download has been applied successfully, or to trigger business logic that depends on the download being confirmed as applied.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.last_download	TIMESTAMP. This is the last download time for the current synchronization.	2

Remarks

This event lets you record the time when the download was successfully applied at the remote database.

This event is only called when using download acknowledgement. The download transaction is committed and the synchronization ends when the download is sent. This event is called when the synchronization client acknowledges a successful download. This event is called on a new connection, after the end_synchronization script of the original synchronization. The actions of this event are committed along with an update to the download time in the MobiLink system tables.

Due to the special nature of this script, any connection-level variables set during the synchronization are not available when this event is executed.

Note
If the download is unsuccessful or if the network connection is dropped, there is no acknowledgement and this script is not invoked. If timely download acknowledgement is critical to your business needs, you should use the last_download parameter of the prepare_for_download script or the last_publication_download parameter of the begin_publication script as backups for your download acknowledgement processing.

See also

- "publication_nonblocking_download_ack connection event" on page 414
- dbmlsync: "SendDownloadAck (sa) extended option" [*MobiLink - Client Administration*]
- UltraLite: "Send Download Acknowledgement synchronization parameter" [*UltraLite - Database Management and Reference*]

SQL example

The following script adds a record to the table download_pubs_acked. The record contains the remote ID, the first authentication parameter, and the download timestamp.

```
INSERT INTO download_pubs_acked( rem_id, auth_parm, last_download )
   VALUES( {ml s.remote_id},  {ml a.1}, {ml s.last_download} )
```

Java example

The following call to a MobiLink system procedure registers a Java method called confirmDownload as the script for the nonblocking_download_ack event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1',
   'nonblocking_download_ack',
   'ExamplePackage.ExampleClass.confirmDownload' )
```

The following is the sample Java method confirmDownload. It calls a Java method to perform business logic based on the download being confirmed, up to the given timestamp, for the given user.

```
public String confirmDownload(
  String user,
  Timestamp ts ) {
  actOnConfirmedDownloadTSForUser( _syncConn, user, ts );
  return( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called ConfirmDownload as the script for the nonblocking_download_ack connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'nonblocking_downlolad_ack',
 'TestScripts.Test.ConfirmDownload'
)
```

The following is the sample .NET method ConfirmDownload. It calls a .NET method to perform business logic based on the download being confirmed, up to the given timestamp, for the given user.

```
public string ConfirmDownload(
  string user,
  DateTime ts ) {
  ActOnConfirmedDownloadTSForUser ( _syncConn, user, ts );
  return ( null );
}
```

prepare_for_download connection event

Processes any required operations between the upload and download transactions.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.last_download	TIMESTAMP. The earliest last download time of any synchronized table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2

Default action

None.

Remarks

The MobiLink server executes this script in a separate transaction, between the upload transaction and the start of the download transaction.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “end_upload connection event” on page 372
- “begin_download connection event” on page 325
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]
- “Using last download times in scripts” on page 88

SQL example

The following call to a MobiLink system procedure registers a SQL stored procedure called prepareForDownload as the script for the prepare_for_download event when synchronizing the script version ver1.

```
CALL ml_add_connection_script(
   'ver1',
   'prepare_for_download',
   'CALL prepareForDownload(
       { ml s.username } )' )
```

The following is the sample SQL method prepareForDownload. This stored procedure prepares downloads for two tables. For example, it could take information from many tables and store it in temporary tables referenced by the download_cursor scripts for tables T1 and T2.

```
CREATE PROCEDURE prepareForDownload (
  IN ts TIMESTAMP,
  IN "user" VARCHAR(128))
BEGIN
  CALL prepareT1Download( user, ts );
  CALL prepareT2Download( user, ts );
END;
```

Java example

The following call to a MobiLink system procedure registers a Java method called prepareForDownload as the script for the prepare_for_download event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1',
   'prepare_for_download',
   'ExamplePackage.ExampleClass.prepareForDownload' )
```

The following is the sample Java method prepareForDownload. This method prepares downloads for two tables. For example, it could take information from many tables, plus other information accessible from Java, and store it in temporary tables referenced by the download_cursor scripts for tables T1 and T2.

```
public String prepareForDownload(
  Timestamp ts,
  String user ) {
  prepareT1ForDownload( _syncconn, user, ts );
  prepareT2ForDownload( _syncconn, user, ts );
  return( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called PrepareForDownload as the script for the prepare_for_download connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'prepare_for_download',
 'TestScripts.Test.PrepareForDownload'
)
```

The following is the sample .NET method PrepareForDownload. This method prepares downloads for two tables. For example, it could take information from many tables, plus other information accessible from .NET, and store it in temporary tables referenced by the download_cursor scripts for tables T1 and T2.

```
public string PrepareForDownload(
  DateTime ts,
  string user ) {
  PrepareT1ForDownload ( _syncConn, user, ts );
  PrepareT2ForDownload ( _syncConn, user, ts );
  return ( null );
}
```

publication_nonblocking_download_ack connection event

When you use download acknowledgement, this script provides a place to record the information that a publication has been successfully downloaded.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.last_publication_download	TIMESTAMP. The earliest last download time of any synchronized table.	2

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.publication name	VARCHAR(128). The name of the publication.	3
s.subscription_id	VARCHAR(128). The remote subscription ID.	4

Remarks

This event lets you record the time when the download of this publication was successfully applied at the remote database.

This event is only called when using download acknowledgement. When in non-blocking mode, the download transaction is committed and the synchronization ends when the download is sent. When the synchronization client acknowledges a successful download, this event is called once per publication in the download. This event is called on a new connection and after the end_synchronization script of the original synchronization. The actions of this event are committed along with an update to the download time in the MobiLink system tables.

Note
If the download is unsuccessful or if the network connection is dropped, there is no acknowledgement and this script is not invoked. If timely download acknowledgement is critical to your business needs, you should use the last_download parameter of the prepare_for_download script or the last_publication_download parameter of the begin_publication script as backups for your download acknowledgement processing.

Due to the special nature of this script, any connection-level variables set during the synchronization are not available when this event is executed.

See also

- "nonblocking_download_ack connection event" on page 410
- dbmlsync: "SendDownloadAck (sa) extended option" [*MobiLink - Client Administration*]
- UltraLite: "Send Download Acknowledgement synchronization parameter" [*UltraLite - Database Management and Reference*]

SQL example

The following script adds a record to a table called download_pubs_acked. The record contains the publication name, the first authentication parameter, and a download timestamp.

```
INSERT INTO download_pubs_acked( pub_name, auth_parm, last_download )
   VALUES( {ml s.publication_name},  {ml a.1}, {ml
s.last_publication_download} )
```

Java example

The following call to a MobiLink system procedure registers a Java method called publicationDownloadACK as the script for the publication_nonblocking_download_ack connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script (
  'ver1',
  'publication_nonblocking_download_ack',
  'ExamplePackage.ExampleClass.publicationDownloadACK' )
```

The following is the sample Java method publicationDownloadACK. It performs some business logic by acting on the confirmation if a particularly important publication was downloaded.

```
public String publicationDownloadACK(
  String user,
  Timestamp ldt,
  String pubName ) {
  if( pubName.equals( "MyImportantPublication" ) ) {
    actOnConfirmedDownload( user, ldt );
  }
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called EndTableDownload as the script for the end_download table event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'publication_nonblocking_download_ack',
  'TestScripts.Test.EndTableDownload'
)
```

The following is the sample .NET method EndTableDownload. It performs some business logic by acting on the confirmation if a particularly important publication was downloaded.

```
public string EndTableDownload
  string user,
  DateTime ldt,
  string pub_name ) {
  if( pub_name.Equals( "MyImportantPublication" ) ) {
    ActOnConfirmedDownload( user, ldt );
  }
  return ( null );
}
```

report_error connection event

Allows you to log errors and to record the actions selected by the handle_error script.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.action_code	INTEGER. This is an INOUT parameter. This parameter is mandatory.	1
s.error_code	INTEGER. The native DBMS error code.	2
s.error_message	TEXT. The native DBMS error message.	3
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	4
s.table	VARCHAR(128). The table whose script caused the error.	5

Default action

None.

Remarks

This script allows you to log errors and to record the actions selected by the handle_error script. This script is executed after the handle_error event, whether a handle_error script is defined. It is always executed in its own transaction, on a different database connection than the synchronization connection (the administrative/information connection).

The MobiLink server always reports an error if the error is recoverable and the MobiLink server is planning to call the handle_error or handle_odbc_error script. For instance, if an error occurs when the MobiLink server is trying to upload an insert, the MobiLink server reports this error and calls the hande_error script. If the action returned from the handle_script is 1000, then the server ignores the error and continues the synchronization. However, if the MobiLink server detects an error before sending anything to the consolidated database, the server may not report the error because the error is not recoverable. More precisely, the MobiLink server reports the errors generated by the ODBC driver and the consolidated database.

The error code and error message allow you to identify the nature of the error. The action code value is returned by the last call to an error handling script for the SQL operation that caused the current error.

If the error happened as part of synchronization, the user name is supplied. Otherwise, this value is null.

If the error happened while manipulating a particular table, the table name is supplied. Otherwise, this value is null. The table name is the name of a table in the remote database. This name may or may not have a direct counterpart in the consolidated database, depending on how your remote table names map to consolidated database table names.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "handle_error connection event" on page 387
- "handle_odbc_error connection event" on page 391
- "report_odbc_error connection event" on page 419
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

The following example works with a SQL Anywhere consolidated database. It inserts a row into a table used to record synchronization errors.

```
CALL ml_add_connection_script(
 'ver1',
 'report_error',
 'INSERT INTO sync_error(
   action_code,
   error_code,
   error_message,
   user_name,
   table_name )
  VALUES (
   {ml s.action_code},
   {ml s.error_code},
   {ml s.error_message},
   {ml s.username},
   {ml s.table} )'  )
```

Java example

The following call to a MobiLink system procedure registers a Java method called reportError as the script for the report_error connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1',
   'report_error',
   'ExamplePackage.ExampleClass.reportError' )
```

The following is the sample Java method reportError. It logs the error to a table using the JDBC connection provided by MobiLink. It also sets the action code.

```
public String reportError(
  ianywhere.ml.script.InOutInteger actionCode,
  int errorCode,
  String errorMessage,
  String user,
  String table )
  throws java.sql.SQLException {
   // Insert error information in a table,
```



```
    JDBCLogError( _syncConn, errorCode, errorMessage,
     user, table );
    actionCode.setValue( getActionCode( errorCode ) );
    return( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called ReportError as the script for the report_error connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
   'ver1',
   'report_error',
   'TestScripts.Test.ReportError' )
```

The following is the sample .NET method ReportError. It logs the error to a table using a .NET method.

```
public string ReportError(
  ref int actionCode,
  int errorCode,
  string errorMessage,
  string user,
  string table ) {
  LogError(_syncConn, errorCode, errorMessage, user, table);
}
```

report_odbc_error connection event

Allows you to log errors and to record the actions selected by the handle_odbc_error script.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.action_code	INTEGER. This is an INOUT parameter. This parameter is mandatory.	1
s.odbc_state	VARCHAR(5). The ODBC SQLSTATE.	2

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.error_message	TEXT. The ODBC error message.	3
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	4
s.table	VARCHAR(128). The table whose script caused the error.	5

Default action

None.

Remarks

This script allows you to log errors and to record the actions selected by the handle_odbc_error script. This script is executed after the handle_odbc_error event, whether or not a handle_odbc_error script is defined. It is always executed in its own transaction, on a different database connection than the synchronization connection (the administrative/information connection).

The ODBC state and error message allow you to identify the nature of the error. The action code value is returned by the last call to an error handling script for the SQL operation that caused the current error.

If the error happened as part of synchronization, the user name is supplied. Otherwise, this value is null.

If the error happened while manipulating a particular table, the table name is supplied. Otherwise, this value is null. The table name is the name of a table in the remote database. This name may or may not have a direct counterpart in the consolidated database, depending on how your remote table names map to consolidated database table names.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “handle_error connection event” on page 387
- “handle_odbc_error connection event” on page 391
- “report_error connection event” on page 416
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following example works with a SQL Anywhere consolidated database. It inserts a row into a table used to record synchronization errors.

```
CALL ml_add_connection_script(
 'ver1',
 'report_odbc_error',
```



```
 'INSERT INTO sync_error(
    action_code,
    odbc_state,
    error_message,
    user_name,
    table_name )
  VALUES(
   {ml s.action_code},
   {ml s.odbc_state},
   {ml s.error_message},
   {ml s.username},
   {ml s.table} )' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called reportODBCError as the script for the report_odbc_error event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1',
   'report_odbc_error',
   'ExamplePackage.ExampleClass.reportODBCError' )
```

The following is the sample Java method reportODBCError. It logs the error to a table using the JDBC connection provided by MobiLink. It also sets the action code.

```
public String reportODBCError(
  ianywhere.ml.script.InOutInteger actionCode,
  String ODBCState,
  String errorMessage,
  String user,
  String table )
  throws java.sql.SQLException {
  JDBCLogError( _syncConn, ODBCState, errorMessage,
   user, table );
  actionCode.setValue( getActionCode( ODBCState ) );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called ReportODBCError as the script for the report_odbc_error event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
   'ver1',
   'report_odbc_error',
   'TestScripts.Test.ReportODBCError' )
```

The following is the sample .NET method ReportODBCError. It logs the error to a table using a .NET method.

```
public string ReportODBCError (
  ref int actionCode,
  string ODBCState,
  string errorMessage,
  string user,
  string table ) {
  LogError(_syncConn, ODBCState, errorMessage, user, table);
  return ( null );
}
```

resolve_conflict table event

Defines a process for resolving a conflict in a specific table.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2

Default action

None.

Remarks

When a row is updated on a remote database, the MobiLink client saves a copy of the original values. The client sends both old and new values to the MobiLink server.

When the MobiLink server receives an updated row, it compares the original values with the present values in the consolidated database. The comparison is done using the upload_fetch script.

If the old uploaded values do not match the current values in the consolidated database, the row conflicts. Instead of updating the row, the MobiLink server inserts both old and new values into the consolidated database. The old and new rows are handled using the upload_old_row_insert and upload_new_row_insert scripts, respectively.

Once the values have been inserted, the MobiLink server executes the resolve_conflict script. It provides the opportunity to resolve the conflict. You can implement any scheme of your choosing.

This script is executed once per conflict.

Alternatively, instead of defining the resolve_conflict script, you can resolve conflicts in a set-oriented fashion by putting conflict-resolution logic either in your end_upload_rows script or in your end_upload table script.

You can have one resolve_conflict script for each table in the remote database.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "upload_old_row_insert table event" on page 507
- "upload_new_row_insert table event" on page 505
- "upload_update table event" on page 519
- "end_upload_rows table event" on page 379
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

The following statement defines a resolve_conflict script suited to the CustDB sample application for an Oracle installation. It calls a stored procedure ULResolveOrderConflict.

```
exec ml_add_table_script(
   'custdb', 'ULOrder', 'resolve_conflict',
   'begin ULResolveOrderConflict();
end; ')
CREATE OR REPLACE PROCEDURE ULResolveOrderConflict()
AS
  new_order_id integer;
  new_status   varchar(20);
  new_notes   varchar(50);
BEGIN
  -- approval overrides denial
  SELECT order_id, status, notes
    INTO new_order_id, new_status, new_notes
    FROM ULNewOrder
   WHERE syncuser_id = SyncUserID;
  IF new_status = 'Approved' THEN
    UPDATE ULOrder o
       SET o.status = new_status, o.notes =
        new_notes
     WHERE o.order_id = new_order_id;
  END IF;
  DELETE FROM ULOldOrder;
  DELETE FROM ULNewOrder;
END;
```

Java example

The following call to a MobiLink system procedure registers a Java method called resolveConflict as the script for the resolve_conflict table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
   'ver1',
   'table1',
   'resolve_conflict',
   'ExamplePackage.ExampleClass.resolveConflict' )
```

The following is the sample Java method resolveConflict. It calls a Java method that uses the JDBC connection provided by MobiLink to resolve the conflict.

```
public String resolveConflict(
  String user,
  String table) {
  resolveRows(_syncConn, user );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called ResolveConflict as the script for the resolve_conflict table event when synchronizing the script version ver1.

```
CALL ml_add_dnet_table_script(
   'ver1',
   'table1',
   'resolve_conflict',
   'TestScripts.Test.ResolveConflict' )
```

The following is the sample .NET method ResolveConflict. It calls a .NET method that resolves the conflict.

```
public string ResolveConflict(
  String user,
  String table) {
  ResolveRows(_syncConn, user );
}
```

synchronization_statistics connection event

Tracks synchronization statistics.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.username	VARCHAR(128). The MobiLink user name.	1
s.warnings	INTEGER. The number of warnings issued during the synchronization.	2
s.errors	INTEGER. The number of errors that occurred during the synchronization.	3
s.deadlocks	INTEGER. The number of deadlocks in the consolidated database that were detected for the synchronization.	4
s.synchronized_tables	INTEGER. The number of client tables that were involved in the synchronization.	5
s.connection_retries	INTEGER. The number of times the MobiLink server retried the connection to the consolidated database.	6

Default action

None.

Remarks

The synchronization_statistics event allows you to gather, for any user and connection, various statistics about the current synchronization. The synchronization_statistics connection script is called just before the commit at the end of the end synchronization transaction.

Note
Depending on the command line, not all warnings are logged. The warnings count passed to this script is the number of warnings that would be logged when no warnings are disabled, which may be more than the number of warnings logged.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “download_statistics connection event” on page 351
- “download_statistics table event” on page 354
- “upload_statistics connection event” on page 509
- “upload_statistics table event” on page 514
- “synchronization_statistics table event” on page 427
- “time_statistics connection event” on page 430
- “time_statistics table event” on page 433
- “MobiLink Monitor” on page 154
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following example inserts synchronization statistics into the sync_con_audit table.

```
CALL ml_add_connection_script(
  'ver1',
  'synchronization_statistics',
 'INSERT INTO sync_con_audit(
   ml_user,
   warnings,
   errors,
   deadlocks,
   synchronized_tables,
   connection_retries)
  VALUES (
   {ml s.username},
   {ml s.warnings},
   {ml s.errors},
   {ml s.deadlocks},
   {ml s.synchronized_tables},
   {ml s.connection_retries})' )
```

Once statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following call to a MobiLink system procedure registers a Java method called synchronizationStatisticsConnection as the script for the synchronization_statistics connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
  'ver1',
  'synchronization_statistics',
  'ExamplePackage.ExampleClass.synchronizationStatisticsConnection'
)
```

The following is the sample Java method synchronizationStatisticsConnection. It logs some of the statistics to the MobiLink message log. (Note that logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String synchronizationStatisticsConnection(
  String user,
```



```
  int warnings,
  int errors,
  int deadlocks,
  int synchronizedTables,
  int connectionRetries ) {
  java.lang.System.out.println(
   "synch statistics number of deadlocks: "
   + deadlocks ;
  return( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called SyncStats as the script for the synchronization_statistics connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'synchronization_statistics',
  'TestScripts.Test.SyncStats'
)
```

The following is the sample .NET method SyncStats. It logs some of the statistics to the MobiLink message log. (Note that logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string SyncStats(
  string user,
  int warnings,
  int errors,
  int deadLocks,
  int syncedTables,
  int connRetries ) {
  System.Console.WriteLine( "synch statistics
   number of deadlocks: " + deadlocks ;
  return( null );
}
```

synchronization_statistics table event

Provides access to synchronization statistics.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.warnings	INTEGER. The number of warnings that occurred for the table during the synchronization.	3
s.errors	INTEGER. The number of errors that were related to the table during the synchronization.	4

Default action

None.

Remarks

The synchronization_statistics event allows you to gather, for any user and table, the number of warnings and errors that occurred during synchronization. The synchronization_statistics table script is called just before the commit at the end of the end synchronization transaction.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “download_statistics connection event” on page 351
- “download_statistics table event” on page 354
- “upload_statistics connection event” on page 509
- “upload_statistics table event” on page 514
- “synchronization_statistics connection event” on page 424
- “time_statistics connection event” on page 430
- “time_statistics table event” on page 433
- “MobiLink Monitor” on page 154
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following example inserts synchronization statistics into the sync_tab_audit table.

```
CALL ml_add_table_script(
  'ver1',
  'table1',
```



```
  'upload_insert',
  'INSERT INTO sync_tab_audit (
    ml_user,
    table,
    warnings,
    errors)
   VALUES (
    {ml s.username},
    {ml s.table},
    {ml s.warnings},
    {ml s.errors} ) ' )
```

Once synchronization statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following call to a MobiLink system procedure registers a Java method called synchronizationStatisticsTable as the script for the synchronization_statistics table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'synchronization_statistics',
  'ExamplePackage.ExampleClass.synchronizationStatisticsTable'
)
```

The following is the sample Java method synchronizationStatisticsTable. It logs some of the statistics to the MobiLink message log. (Note that logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String synchronizationStatisticsTable(
  String user,
  String table,
  int warnings,
  int errors ) {
  java.lang.System.out.println( "synch statistics for
   table: " + table + " errors: " + errors );
  return( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called SyncTableStats as the script for the synchronization_statistics table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'synchronization_statistics',
  'TestScripts.Test.SyncTableStats'
)
```

The following is the sample .NET method SyncTableStats. It logs some of the statistics to the MobiLink message log. (Note that logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string SyncTableStats(
  string user,
  string table,
  int warnings,
  int errors ) {
  System.Console.WriteLine( "synch statistics for
   table: " + table + " errors: " + errors );
  return( null );
}
```

time_statistics connection event

Tracks time statistics by user and event.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.event_name	VARCHAR(128). The event whose statistics are being reported.	2
s.number_of_calls	INTEGER. The number of times the script was called.	3
s.minimum_time	INTEGER. Milliseconds. The shortest time it took to execute a script during this synchronization.	4

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.maximum_time	INTEGER. Milliseconds. The longest time it took to execute a script during this synchronization.	5
s.total_time	INTEGER. Milliseconds. The total time it took to execute all scripts in the synchronization. (This is not the same as the length of the synchronization.)	6

Default action

None.

Remarks

The time_statistics event allows you to gather time statistics for a synchronization. The statistics are gathered only for those events for which there is a corresponding script. The script gathers aggregate data for occasions where a single event occurs multiple times.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “time_statistics table event” on page 433
- “download_statistics connection event” on page 351
- “download_statistics table event” on page 354
- “upload_statistics connection event” on page 509
- “upload_statistics table event” on page 514
- “synchronization_statistics connection event” on page 424
- “synchronization_statistics table event” on page 427
- “MobiLink Monitor” on page 154
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following example inserts statistical information into the time_statistics table.

```
CALL ml_add_connection_script(
  'ver1',
  'time_statistics',
  'INSERT INTO time_statistics (
    id,
    ml_user,
    event_name,
    number_of_calls,
    minimum_time,
    maximum_time,
    total_time)
   VALUES (
    ts_id.nextval,
```

```
        {ml s.username},
        {ml s.event_name},
        {ml s.number_of_calls},
        {ml s.minimum_time},
        {ml s.maximum_time},
        {ml s.total_time} ) ' )
```

Java example

The following call to a MobiLink system procedure registers a Java method called timeStatisticsConnection as the script for the time_statistics connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
  'ver1',
  'time_statistics',
  'ExamplePackage.ExampleClass.timeStatisticsConnection' )
```

The following is the sample Java method timeStatisticsConnection. It prints statistics for the prepare_for_download event. (Note that printing statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String timeStatisticsConnection(
  String username,
  String tableName,
  String eventName,
  int numberOfCalls,
  int minimumTime,
  int maximumTime,
  int totalTime ) {
  if( eventName.equals( "prepare_for_download") ) {
    java.lang.System.out.println(
     "prepare_for_download num_calls: " + numCalls +
     "total_time: " + totalTime );
  }
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called TimeStats as the script for the time_statistics connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'time_statistics',
  'TestScripts.Test.TimeStats'
)
```

The following is the sample .NET method TimeStats. It prints statistics for the prepare_for_download event. (Note that printing statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string TimeStats(
  string user,
  string eventName,
  int numberOfCalls,
  int minimumTime,
  int maximumTime,
  int totTime ) {
```



```
  if( event_name=="prepare_for_download") {
    System.Console.WriteLine(
     "prepare_for_download num_calls: " + num_calls +
     "total_time: " + total_time );
  }
  return ( null );
}
```

time_statistics table event

Tracks time statistics.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.event_name	VARCHAR(128). The event whose statistics are being reported.	3
s.number_of_calls	INTEGER. The number of times the script was called.	4
s.minimum_time	INTEGER. Milliseconds. The shortest time it took to execute a script during the synchronization of this table.	5

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.maximum_time	INTEGER. Milliseconds. The longest time it took to execute a script during the synchronization of this table.	6
s.total_time	INTEGER. Milliseconds. The total time it took to execute all scripts in the synchronization of the table. (This is not the same as the length of the synchronization.)	7

Default action

None.

Remarks

The time_statistics table event allows you to gather time statistics for a table during synchronization. The statistics are gathered only for those events for which there is a corresponding script. The script gathers aggregate data for occasions where a single event occurs multiple times.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “time_statistics connection event” on page 430
- “download_statistics connection event” on page 351
- “download_statistics table event” on page 354
- “upload_statistics connection event” on page 509
- “upload_statistics table event” on page 514
- “synchronization_statistics connection event” on page 424
- “synchronization_statistics table event” on page 427
- “MobiLink Monitor” on page 154
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

The following example inserts statistical information into the time_statistics table.

```
CALL ml_add_table_script (
 'ver1',
 'table1',
 'time_statistics',
 'INSERT INTO time_statistics(
   ml_user,
   table,
   event_name,
   number_of_calls,
   minimum_time,
   maximum_time,
```



```
    total_time)
  VALUES (
    {ml s.username},
    {ml s.table},
    {ml s.event_name},
    {ml s.number_of_calls},
    {ml s.minimum_time},
    {ml s.maximum_time},
    {ml s.total_time} )' );
```

Java example

The following call to a MobiLink system procedure registers a Java method called timeStatisticsTable as the script for the time_statistics table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'time_statistics',
  'ExamplePackage.ExampleClass.timeStatisticsTable' )
```

The following is the sample Java method timeStatisticsTable. It prints statistics for the upload_old_row_insert event.

```
public String timeStatisticsTable(
  String username,
  String tableName,
  String eventName,
  int numberOfCalls,
  int minimumTime,
  int maximumTime,
  int totalTime ) {
  if( eventName.equals( "upload_old_row_insert") ) {
    java.lang.System.out.println(
     "upload_old_row_insert num_calls: " + numCalls +
     "total_time: " + totalTime );
  }
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called TimeTableStats as the script for the time_statistics table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'time_statistics',
  'TestScripts.Test.TimeTableStats'
)
```

The following is the sample .NET method TimeTableStats. It prints statistics for the upload_old_row_insert event.

```
public string TimeTableStats(
  string user,
  string table,
  string eventName,
  int numberOfCalls,
  int minimumTime,
```

```
    int maximumTime,
    int totTime ) {
    if( event_name == "upload_old_row_insert") {
      System.Console.WriteLine(
        "upload_old_row_insert num_calls: " + num_calls +
        "total_time: " + total_time );
    }
    return ( null );
  }
```

upload_delete table event

A data script that provides an event that the MobiLink server uses during processing of the upload to handle rows deleted from the remote database.

Parameters

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description Order (deprecated for SQL)
s.*remote_id*	VARCHAR(128). The MobiLink remote ID. You

Parameter name for SQL scripts	DOrder (deprecated for SQL) escription
	canonlyreferencetheremoteIDifyouarereus

Parameter name for SQL scripts	Description
	ingnamedparameters.

Parameter name for SQL scripts	Description Order (deprecated for SQL)
s.*username*	Optional ARCHAR(128). The MobiLink user name. This

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
	sparameterisoptional.	

Parameter name for SQL scripts	Description (deprecated for SQL)
r. *pk-column-1*	Required. The first deleted primary key co

Parameter name for SQL scripts	Description Order (deprecated for SQL)
	lumnvalue, referencedbycolumnnameorcol

Parameter name for SQL scripts	Description Order (deprecated for SQL)
	number.
...	...

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
r. *pk-column-N*	Required. The last deleted primary key column	RN

Parameter name for SQL scripts	Description
	umnvalue, referencedbycolumnnameorcolu

Parameter name for SQL scripts	Description Order (deprecated for SQL)
	mnnumber.

Parameter name for SQL scripts	Description
r. *column-1*	RN + 1 equired. The first deleted non-primary k

Parameter name for SQL scripts	Description Order (deprecated for SQL)
	eycolumnvalue, referencedbycolumnnameo

Parameter name for SQL scripts	Description Order (deprecated for SQL)
	r c o l u m n n u m b e r .
...	. ... . .

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
r. *column-M*	required. The last deleted non-primary key	*N* + *M*

Parameter name for SQL scripts	Description
	y c o l u m n v a l u e , r e f e r e n c e d b y c o l u m n n a m e o r

Parameter name for SQL scripts	Description Order (deprecated for SQL) column number.

Default action

None.

Remarks

The action taken at the consolidated database can be a DELETE statement, but need not be.

You can have one upload_delete script for each table in the remote database.

This script must be implemented in SQL. For Java or .NET processing of rows, see “Direct row handling” on page 671.

Note
Conflict detection is usually performed much faster when done all at once in the upload_update script. See “upload_update table event” on page 519.

See also

- “Data scripts” on page 308
- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “upload_insert table event” on page 487
- “upload_update table event” on page 519

SQL example

This example is taken from the Contact sample and can be found in *Samples\MobiLink\Contact \build_consol.sql*. It marks customers that are deleted from the remote database as inactive.

```
CALL ml_add_table_script(
  'ver1',
  'Customer',
  'upload_delete',
  'UPDATE Customer
   SET active = 0
   WHERE cust_id={ml r.cust_id}' )
```

upload_fetch table event

A data script that fetches rows from a synchronized table in the consolidated database for row-level conflict detection.

Parameters

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You	N/A

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
	can only reference the remote ID if you are using	

Parameter name for SQL scripts	Description
	ingnamedparameters.

Parameter name for SQL scripts	Description Order (deprecated for SQL)
s.*username*	Optional ARCHAR(128). The MobiLink user name. This

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
	sparameteriisoptional.	

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
r. *primary-key-1*	Required. The first primary key column value	1 (2 if username is referenced)

Parameter name for SQL scripts	Description (deprecated for SQL)
	ue, referenced by column name or column number

Parameter name for SQL scripts	DOrder (deprecated for SQL) e s c r i p t i o n
	e r .

Parameter name for SQL scripts	Description (deprecated for SQL)
r. *primary-key-2*	Required. The second primary key column value

Parameter name for SQL scripts	Description
	value, referenced by column name or column number

Parameter name for SQL scripts	Description
	Order (deprecated for SQL)
	b e r .
...	. ... . .

Parameter name for SQL scripts	Description Order (deprecated for SQL)
r. *primary-key-N*	RequiredN (N+1 if username is referenced) The last primary key column value

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
	e, referenced by column name or column number	

Parameter name for SQL scripts	Description
	Order (deprecated for SQL) r .

Default action

None.

Remarks

The upload_fetch script is a companion to the upload_update event.

The columns of the result set must match the number and order of columns being uploaded from the remote database for this table. If the values returned do not match the pre-image in the uploaded row, a conflict is identified.

Do not use READPAST table hints in upload_fetch scripts. If the script skips a locked row using READPAST, the synchronization logic thinks that the row was deleted. Depending on what scripts you have defined, this either causes the uploaded update to be ignored or it triggers conflict resolution. Ignoring the update is likely to be unacceptable behavior, and may be harmful. Triggering conflict resolution may not be a problem, depending on the resolution logic you have implemented.

You can have only one upload_fetch or upload_fetch_column_conflict script for each table in the remote database.

This script must be implemented in SQL. For Java or .NET processing of rows, see “Direct row handling” on page 671.

This script may be ignored if none of the following scripts are defined: upload_new_row_insert, upload_old_row_insert, and resolve_conflict.

See also

- “Data scripts” on page 308
- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “Detecting conflicts” on page 103
- “resolve_conflict table event” on page 422
- “upload_delete table event” on page 436
- “upload_insert table event” on page 487
- “upload_update table event” on page 519
- "Using READPAST with MobiLink synchronization" in “FROM clause” [*SQL Anywhere Server - SQL Reference*]

SQL example

The following SQL script is taken from the Contact sample and can be found in *samples-dir\MobiLink\Contact\build_consol.sql*. It is used to identify conflicts that occur when rows updated in the remote database Product table are uploaded. This script selects rows from a table also named Product, but depending on your consolidated and remote database schema, the two table names may not match.

```
CALL ml_add_table_script(
  'ver1',
  'Product',
  'upload_fetch',
  'SELECT id, name, size, quantity, unit_price
   FROM Product
   WHERE id={ml r.id}' )
```

upload_fetch_column_conflict table event

A data script that fetches rows from a synchronized table in the consolidated database for column-level conflict detection.

Parameters

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description Order (deprecated for SQL)
s.*remote_id*	VARCHAR(128). The MobiLink remote ID. You

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
	can only reference the remote ID if you are using

Parameter name for SQL scripts	Description
	ingnamedparameters.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.*username*	Optional ARCHAR(128). The MobiLink username. This	V

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
	sparameteriso ptional.	

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
r. *pk-column-1*	Required. The first primary key column value	1 (2 if username is referenced)

Parameter name for SQL scripts	Description
	Order (deprecated for SQL)
	True, reference by column name or column number

Parameter name for SQL scripts	Description
	er.
...	....

Parameter name for SQL scripts	Description Order (deprecated for SQL)
r. *pk-column-N*	RN (*N*+1 if username is referenced) required. The last primary key column value

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
	e, referenced by column name or column number	

Parameter name for SQL scripts	Description
	Order (deprecated for SQL) r .

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
r. *column-1*	Required. The first non-primary key column	N + 1 (*N*+2 if username is referenced)

Parameter name for SQL scripts	Description
	mnvalue, referenced by column name or column

Parameter name for SQL scripts	Order (deprecated for SQL) description
	number.
...	...

Parameter name for SQL scripts	Description (deprecated for SQL)
r. *column-M*	RN + M (N+M+1 if username is required). The last non-primary key column

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
	value, referenced by column name or column n	

Parameter name for SQL scripts	DOrder (deprecated for SQL) e s c r i p t i o n
	u m b e r .

Default action

None.

Remarks

The upload_fetch_column_conflict script is a companion to the upload_update event.

This script can only be defined for remote tables that have no BLOBs.

With this script, the MobiLink server only detects a conflict for a row when the same column was updated on the remote database and the consolidated database since the last synchronization. Different users can update the same row without generating a conflict, as long as they don't update the same column.

For example, using the upload_fetch_column_conflict script, you could avoid detecting a conflict when one of your remote users updated the quant column of the ULOrder table, and another remote user updated the notes column for the same row. You would only detect a conflict if they both updated the quant column.

Note
Conflict detection is usually performed much faster when done all at once in the upload_update script. See “upload_update table event” on page 519.

When using an upload_fetch_column_conflict script and no conflict is detected, the row values passed to your upload_update script come from either the remote database's upload or the current consolidated values from your upload_fetch_column_conflict script. The remote database's value is used for columns that were updated on the remote database, otherwise the current consolidated value is used. In other words, only the columns that were updated on the remote are updated in the consolidated.

You can have only one upload_fetch or upload_fetch_column_conflict script for each table in the remote database.

This script may be ignored if none of the following scripts are defined: upload_new_row_insert, upload_old_row_insert, and resolve_conflict.

This script must be implemented in SQL. For Java or .NET processing of rows, see "Direct row handling" on page 671.

See also

- "Data scripts" on page 308
- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "Detecting conflicts" on page 103
- "upload_fetch table event" on page 454
- "resolve_conflict table event" on page 422
- "upload_delete table event" on page 436
- "upload_insert table event" on page 487
- "upload_update table event" on page 519

upload_insert table event

A data script that provides an event that the MobiLink server uses during processing of the upload to handle rows inserted into the remote database.

Parameters

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.*remote_id*	VARCHAR(128). The MobiLink remote ID. You	N/A

Parameter name for SQL scripts	Description
	Order (deprecated for SQL) can only reference the remote ID if you are using

Parameter name for SQL scripts	Description
	ing named parameters.

Parameter name for SQL scripts	Description (deprecated for SQL)
s.username	VARCHAR(128). The MobiLink username. This

Parameter name for SQL scripts	DOrder (deprecated for SQL) e s c r i p t i o n
	s p a r a m e t e r i s o p t i o n a l .

Parameter name for SQL scripts	Description (deprecated for SQL)
r. *pk-column-1*	Required. The first primary key c

Parameter name for SQL scripts	DOrder (deprecated for SQL) escription
	olumnvalue, referencedbycolumnnameorco

Parameter name for SQL scripts	Order (deprecated for SQL) D e s c r i p t i o n
	l u m n v a l u e .
...	. ... . .

Parameter name for SQL scripts	DOrder (deprecated for SQL) escription
r. *pk-column-N*	RNequired. The last inserted primary key co

Parameter name for SQL scripts	Description Order (deprecated for SQL) lumnvalue, referencedbycolumnnameorcol

Parameter name for SQL scripts	Description
	Order (deprecated for SQL) number value.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
r. *column-1*	Required. The first inserted non-primary	*N*+1

Parameter name for SQL scripts	DOrder (deprecated for SQL) escription
	key column value, reference by column name

Parameter name for SQL scripts	Order (deprecated for SQL) Description
	orcolumnvalue.
...	.... .

Parameter name for SQL scripts	DOrder (deprecated for SQL) escription
r. *column-M*	RN+*M* equired. The last inserted non-primary k

Parameter name for SQL scripts	Description
	Order (deprecated for SQL) eycolumnvalue, referencedbycolumnnameo

Parameter name for SQL scripts	DOrder (deprecated for SQL) e s c r i p t i o n
	r c o l u m n v a l u e .

Default action

None.

Remarks

You can have one upload_insert script for each table in the remote database.

This script must be implemented in SQL. For Java or .NET processing of rows, see "Direct row handling" on page 671.

See also

- "Data scripts" on page 308
- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "upload_delete table event" on page 436
- "upload_update table event" on page 519
- "upload_fetch table event" on page 454

SQL example

This example handles inserts that were made on the Customer table in the remote database. The script inserts the values into a table named Customer in the consolidated database. The final column of the table identifies the Customer as active. The final column does not appear in the remote database.


```
CALL ml_add_table_script(
 'ver1',
 'Customer',
 'upload_insert',
 'INSERT INTO Customer(
   cust_id,
   name,
   rep_id,
   active )
  VALUES (
   {ml r.cust_id},
   {ml r.name},
   {ml r.rep_id},
   1 )' );
```

upload_new_row_insert table event

Conflict resolution scripts for statement-based uploads commonly require access to the old and new values of rows uploaded from the remote database. This data script event allows you to handle the new, updated values of rows uploaded from the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name. This parameter is optional.	Optional (1 if referenced)
r. *pk-column-1*	Required. The first primary key column value from the new (post-image) row, referenced by column name or column number.	1 (2 if username is referenced)
...	...	...
r. *pk-column-N*	Required. The last primary key column value from the new (post-image) row, referenced by column name or column number.	*N* (*N*+1 if username is referenced)

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
r. *column-1*	Required. The first non-primary key column value from the new (post-image) row, referenced by column name or column number.	*N* + 1 (*N*+2 if user-name is referenced)
...	...	...
r. *column-M*	Required. The last non-primary key column value from the new (post-image) row, referenced by column name or column number.	*N* + *M* (*N*+*M*+1 if username is referenced)

Default action

None.

Remarks

When a MobiLink client sends an updated row to the MobiLink server, it includes not only the new values (the post-image), but also a copy of the old row values (the pre-image). When the pre-image does not match the current values in the consolidated database, a conflict is detected.

After MobiLink detects a conflict, this event allows you to save post-image values to a table. You can use this event to assist in developing conflict resolution procedures for updates. The parameters for this event hold new row values from the remote database before the update is performed on the corresponding consolidated database table. This event is also used to insert rows in forced-conflict mode (forced-conflict mode has been deprecated.).

Note
Conflict detection is usually performed much faster when done all at once in the upload_update script. See "upload_update table event" on page 519.

The script for this event is usually an insert statement that inserts the new row into a temporary table for use by a resolve_conflict script.

You can have one upload_new_row_insert script for each table in the remote database.

This script must be implemented in SQL. For Java or .NET processing of rows, see "Direct row handling" on page 671.

See also

- “Data scripts” on page 308
- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “Handling conflicts” on page 102
- “resolve_conflict table event” on page 422
- “upload_old_row_insert table event” on page 507
- “upload_update table event” on page 519
- “Forced conflicts (Deprecated)” on page 111
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL example

This example handles updates made on the product table in the remote database. The script inserts the new value of the row into a global temporary table named product_conflict. The final column of the table identifies the row as a new row.

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'upload_new_row_insert',
 'INSERT INTO DBA.product_conflict(
   id,
   name,
   size,
   quantity,
   unit_price,
   row_type )
  VALUES(
   {ml r.id},
   {ml r.name},
   {ml r.size},
   {ml r.quantity},
   {ml r.unit_price},
   ''New'' )' )
```

upload_old_row_insert table event

Conflict resolution scripts for statement-based uploads commonly require access to the old and new values of rows uploaded from the remote database. This data script event allows you to handle the old values of rows uploaded from the remote database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See “SQL-Java data types” on page 526 and “SQL-.NET data types” on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use

parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	N/A
s.username	VARCHAR(128). The MobiLink user name. This parameter is optional.	Optional (1 if referenced)
r. *pk-column-1*	Required. The first primary key column value from the old (pre-image) row, referenced by column name or column number.	1 (2 if username is referenced)
...	...	...
r. *pk-column-N*	Required. The last primary key column value from the old (pre-image) row, referenced by column name or column number.	*N* (*N*+1 if username is referenced)
r. *column-1*	Required. The first non-primary key column value from the old (pre-image) row, referenced by column name or column number.	*N* + 1 (*N*+2 if username is referenced)
...	...	...
r. *column-M*	Required. The last non- primary key column value from the old (pre-image) row, referenced by column name or column number.	*N* + *M* (*N*+*M*+1 if username is referenced)

Default action

None.

Remarks

When a MobiLink client sends an updated row to the MobiLink server, it includes not only the new values (the post-image), but also a copy of the old row values (the pre-image). When the pre-image does not match the current values in the consolidated database, a conflict is detected.

After MobiLink detects a conflict, this event allows you to save pre-image values to a table. You can use this event to assist in developing conflict resolution procedures. The parameters for this event hold old row values from the remote database before the update is performed on the corresponding consolidated database table. This event is also used to insert rows in forced-conflict mode (forced-conflict mode has been deprecated).

Note
Conflict detection is usually much faster when done all at once in the upload_update script. See "upload_update table event" on page 519.

The script for this event is usually an insert statement that inserts the old row into a temporary table for use by a resolve_conflict script.

You can have one upload_old_row_insert script for each table in the remote database.

This script must be implemented in SQL. For Java or .NET processing of rows, see "Direct row handling" on page 671.

See also

- "Data scripts" on page 308
- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "Handling conflicts" on page 102
- "resolve_conflict table event" on page 422
- "upload_new_row_insert table event" on page 505
- "upload_update table event" on page 519
- "Forced conflicts (Deprecated)" on page 111
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

This example handles updates made on the product table in the remote database. The script inserts the old value of the row into a global temporary table named product_conflict. The final column of the table identifies the row as an old row.

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'upload_old_row_insert',
 'INSERT INTO DBA.product_conflict (
   id,
   name,
   size,
   quantity,
   unit_price,
   row_type )
  VALUES (
   {ml r.id},
   {ml r.name},
   {ml r.size},
   {ml r.quantity},
   {ml r.unit_price},
   ''Old'' )' )
```

upload_statistics connection event

Provides access to synchronization statistics for upload operations.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.warnings	INTEGER. The number of warnings that occurred.	2
s.errors	INTEGER. The number of errors that occurred.	3
s.inserted_rows	INTEGER. The number of rows that were successfully inserted in the consolidated database.	4
s.deleted_rows	INTEGER. The number of rows that were successfully deleted from the consolidated database.	5
s.updated_rows	INTEGER. The number of rows that were successfully updated in the consolidated database.	6
s.conflicted_inserts	INTEGER. Always zero.	7
s.conflicted_deletes	INTEGER. Always zero.	8
s.conflicted_updates	INTEGER. The number of update rows that caused conflict. A row is included only when a resolve conflict script was successfully called for it.	9

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.ignored_inserts	INTEGER. The total number of upload insert rows that were ignored. They were ignored because 1) there is no upload_insert script in normal mode or no upload_new_row_insert script in forced conflict mode; or 2) errors occurred when the MobiLink server was invoking the corresponding script and the handle_error or handle_odbc_error event returned 1000.	10
s.ignored_deletes	INTEGER. The number of upload delete rows that caused errors while the upload_delete script was invoked, when the handle_error or handle_odbc_error are defined and returned 1000, or when there is no upload_delete script defined for the given table.	11
s.ignored_updates	INTEGER. The number of upload update rows that caused conflict but a resolve conflict script was not successfully called or no upload_update script was defined.	12
s.bytes	INTEGER. The amount of memory used within the MobiLink server to store the upload.	13
s.deadlocks	INTEGER. The number of deadlocks in the consolidated database that were detected for the synchronization.	14

Default action

None.

Remarks

The upload_statistics event allows you to gather, for any user, statistics on uploads. The upload_statistics connection script is called just before the commit at the end of the upload transaction.

See also

- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "download_statistics connection event" on page 351
- "download_statistics table event" on page 354
- "upload_statistics table event" on page 514
- "synchronization_statistics connection event" on page 424
- "synchronization_statistics table event" on page 427
- "time_statistics connection event" on page 430
- "time_statistics table event" on page 433
- "MobiLink Monitor" on page 154
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]

SQL example

The following example inserts synchronization statistics for upload operations into the table upload_summary_audit.

```
CALL ml_add_connection_script (
 'ver1',
 'upload_statistics',
 'INSERT INTO upload_summary_audit (
   ml_user,
   warnings,
   errors,
   inserted_rows,
   deleted_rows,
   updated_rows,
   conflicted_inserts,
   conflicted_deletes,
   conflicted_updates,
   ignored_inserts,
   ignored deletes,
   ignored_updates,
   bytes, deadlocks )
  VALUES (
   {ml s.username},
   {ml s.warnings},
   {ml s.errors},
   {ml s.inserted_rows},
   {ml s.deleted_rows},
   {ml s.updated_rows},
   {ml s.conflicted_inserts},
   {ml s.conflicted_deletes},
   {ml s.conflicted_updates},
   {ml s.ignored_inserts},
   {ml s.ignored_deletes},
   {ml s.ignored_updates},
   {ml s.bytes},
   {ml s.deadlocks} ) ' )
```

Once statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following call to a MobiLink system procedure registers a Java method called uploadStatisticsConnection as the script for the upload_statistics connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
   'ver1',
   'upload_statistics',
   'ExamplePackage.ExampleClass.uploadStatisticsConnection' )
```

The following is the sample Java method uploadStatisticsConnection. It logs some statistics to the MobiLink message log. (Note that logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String uploadStatisticsConnection(
  String user,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictedInserts,
  int conflictedDeletes,
  int conflictedUpdates,
  int ignoredInserts,
  int ignoredDeletes,
  int ignoredUpdates,
  int bytes,
  int deadlocks ) {
  java.lang.System.out.println( "updated rows: " +
   updatedRows );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called UploadStats as the script for the upload_statistics connection event when synchronizing the script version ver1.

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'upload_statistics',
  'TestScripts.Test.UploadStats'
)
```

The following is the sample .NET method UploadStats. It logs some statistics to the MobiLink message log. (Note that logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string UploadStats (
  string user,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictInserts,
  int conflictDeletes,
  int conflictUpdates,
```

```
    int ignoredInserts,
    int ignoredDeletes,
    int ignoredUpdates,
    int bytes,
    int deadlocks ) {
    System.Console.WriteLine( "updated rows: " +
      updatedRows );
    return ( null );
  }
```

upload_statistics table event

Provides access to synchronization statistics for upload operations for a specific table.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 526 and "SQL-.NET data types" on page 591.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.warnings	INTEGER. The number of warnings issued in the upload of the table.	3
s.errors	INTEGER. The number of errors, including handled errors, that occurred in the upload of the table.	4

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.inserted_rows	INTEGER. The number of rows that were successfully inserted in the consolidated database.	5
s.deleted_rows	INTEGER. The number of rows that were successfully deleted from the consolidated database.	6
s.updated_rows	INTEGER.	7
s.conflicted_inserts	INTEGER. Always zero.	8
s.conflicted_deletes	INTEGER. Always zero.	9
s.conflicted_updates	INTEGER. The number of update rows that caused conflict. A row is included only when a resolve conflict script was successfully called for it.	10
s.ignored_inserts	INTEGER. The total number of upload insert rows that were ignored. They were ignored because 1) there is no upload_insert script in normal mode or no upload_new_row_insert script in forced conflict mode; or 2) errors occurred when the MobiLink server was invoking the corresponding script and the handle_error or handle_odbc_error event returned 1000.	11
s.ignored_deletes	INTEGER. The number of upload delete rows that caused errors while the upload_delete script was invoked, when the handle_error or handle_odbc_error are defined and returned 1000, or when there is no upload_delete script defined for the given table.	12
s.ignored_updates	INTEGER. The number of upload update rows that caused conflict but a resolve conflict script was not successfully called or no upload_update script was defined.	13

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.bytes	INTEGER. The amount of memory used within the MobiLink server to store the upload.	14
s.deadlocks	INTEGER. The number of deadlocks in the consolidated database that were detected for the synchronization.	15

Default action

None.

Remarks

The upload_statistics event allows you to gather, for any user, vital statistics on synchronization happenings as they apply to any table. The upload_statistics table script is called just before the commit at the end of the upload transaction.

Note
Depending on the command line, not all warnings are logged. The warnings count passed to this script is the number of warnings that would be logged when no warnings are disabled, which may be more than the number of warnings logged.

See also

- “Script parameters” on page 268
- “Adding and deleting scripts” on page 285
- “download_statistics connection event” on page 351
- “upload_statistics connection event” on page 509
- “upload_statistics table event” on page 514
- “synchronization_statistics connection event” on page 424
- “synchronization_statistics table event” on page 427
- “time_statistics connection event” on page 430
- “time_statistics table event” on page 433
- “MobiLink Monitor” on page 154
- “Using remote IDs and MobiLink user names in scripts” [*MobiLink - Client Administration*]

SQL Example

The following example inserts a row into a table used to track upload statistics.

```
CALL ml_add_connection_script(
 'ver1',
 'upload_statistics',
 'INSERT INTO my_upload_statistics (
   user_name,
   table_name,
   num_warnings,
   num_errors,
```



```
    inserted_rows,
    deleted_rows,
    updated_rows,
    conflicted_inserts,
    conflicted_deletes,
    conflicted_updates,
    ignored_inserts,
    ignored_deletes,
    ignored_updates, bytes,
    deadlocks )
   VALUES(
    {ml s.username},
    {ml s.table},
    {ml s.warnings},
    {ml s.errors},
    {ml s.inserted_rows},
    {ml s.deleted_rows},
    {ml s.updated_rows},
    {ml s.conflicted_inserts},
    {ml s.conflicted_deletes},
    {ml s.conflicted_updates},
    {ml s.ignored_inserts},
    {ml s.ignored_deletes},
    {ml s.ignored_updates},
    {ml s.bytes},
    {ml s.deadlocks} )'  )
```

The following example works with an Oracle consolidated database.

```
CALL ml_add_connection_script(
 'ver1',
 'upload_statistics',
 'INSERT INTO upload_tables_audit (
  id,
  user_name,
  table,
  warnings,
  errors,
  inserted_rows,
  deleted_rows,
  updated_rows,
  conflicted_inserts,
  conflicted_deletes,
  conflicted_updates,
  ignored_inserts,
  ignored_deletes,
  ignored_updates,
  bytes,
  deadlocks )
 VALUES (
   ut_audit.nextval,
   {ml s.username},
   {ml s.table},
   {ml s.warnings},
   {ml s.errors},
   {ml s.inserted_rows},
   {ml s.deleted_rows},
   {ml s.updated_rows},
   {ml s.conflicted_inserts},
   {ml s.conflicted_deletes},
   {ml s.conflicted_updates},
   {ml s.ignored_inserts},
   {ml s.ignored_deletes},
```

```
    {ml s.ignored_updates},
    {ml s.bytes},
    {ml s.deadlocks} )' )
```

Once statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following call to a MobiLink system procedure registers a Java method called uploadStatisticsTable as the script for the upload_statistics table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'upload_statistics',
  'ExamplePackage.ExampleClass.uploadStatisticsTable' )
```

The following is the sample Java method uploadStatisticsTable. It logs some statistics to the MobiLink message log. (Note that logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public String uploadStatisticsTable(
  String user,
  String table,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictedInserts,
  int conflictedDeletes,
  int conflictedUpdates,
  int ignoredInserts,
  int ignoredDeletes,
  int ignoredUpdates,
  int bytes,
  int deadlocks ) {
  java.lang.System.out.println( "updated rows: " +
    updatedRows );
  return ( null );
}
```

.NET example

The following call to a MobiLink system procedure registers a .NET method called UploadTableStats as the script for the upload_statistics table event when synchronizing the script version ver1 and the table table1.

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'upload_statistics',
  'TestScripts.Test.UploadTableStats'
)
```

The following is the sample .NET method uploadStatisticsTable. It logs some statistics to the MobiLink message log. (Note that logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
public string UploadTableStats(
  string user,
```



```
    string table,
    int warnings,
    int errors,
    int insertedRows,
    int deletedRows,
    int updatedRows,
    int conflictInserts,
    int conflictDeletes,
    int conflictUpdates,
    int ignoredInserts,
    int ignoredDeletes,
    int ignoredUpdates,
    int bytes,
    int deadlocks ) {
    System.Console.WriteLine( "updated rows: " +
      updatedRows );
    return ( null );
  }
```

upload_update table event

A data script that provides an event that the MobiLink server uses during processing of the upload to handle rows updated at the remote database.

Parameters

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated and it is recommended that you use named parameters. You cannot mix names and question marks within a script. If you use question marks, the parameters must be in the order shown below and are optional only if no subsequent parameters are specified (for example, you must use parameter 1 if you want to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

Parameter	Description	Order (deprecated for SQL)
r. *column-1*	Required. The first non-primary key column value from the new (post-image) column value, referenced by column name or column number.	1
...	...	...
r. *column-M*	Required. The last non-primary key column value from the new (post-image) column value, referenced by column name or column number.	M
r. *pk-column-1*	Required. The first primary key column value from the new (post-image) column value, referenced by column name or column number.	M + 1
...	...	...

Parameter	Description	Order (deprecated for SQL)
r. *pk-column-N*	Required. The last primary key column value from the new (post-image) column value, referenced by column name or column number.	M + N
o. *column-N*	Optional. The first non-primary key column value from the old (pre-image) column value, referenced by column name or column number.	M + N + 1
...	...	...
o. *column-M*	Optional. The last non-primary key column value from the old (pre-image) column value, referenced by column name or column number.	M + N + M

Default action

None.

Remarks

The WHERE clause must include all the primary key columns being synchronized. The SET clause must contain all the non-primary key columns being synchronized.

You can use named parameters in any order. The same named parameter can be used as many times as you want in the same script. You may only specify a subset of the columns in a script with named parameters.

For example, the upload_script for the table MyTable can be written as:

```
UPDATE MyTable
  SET column_2 = { ml r.column_2 }, column_1 = { ml r.column_1 }, ...,
column_M = { ml r.column_M }
  WHERE pk_column_1 = { ml r.pk_column_1 } AND ... AND pk_column_N = { ml
r.pk_column_N }
```

You can have one upload_update script for each table in the remote database.

This script must be implemented in SQL. For Java or .NET processing of rows, see "Direct row handling" on page 671.

To use the upload_update script to detect conflicts, include all non-primary key columns in the WHERE clause:

UPDATE *table-name*
SET *col1* **= {ml r.***col1***},** *col2* **= {ml r.** *col2* **}** ...
WHERE *pk1* **= {ml r.***pk1***} AND** *pk2* **= {ml r.***pk2***}** ...
  AND *col1* **= {ml o.***col1***} AND** *col2* **={ml o.***col2***}** ...

In this statement, col1 and col2 are the non-primary key columns, while pk1 and pk2 are primary key columns. The values passed to the second set of non-primary key columns are the pre-image of the updated row. The WHERE clause compares old values uploaded from the remote to current values in the

consolidated database. If the values do not match, the update is ignored, preserving the values already on the consolidated database.

This script must be implemented in SQL. For Java or .NET processing of rows, see "Direct row handling" on page 671.

See also

- "Data scripts" on page 308
- "Script parameters" on page 268
- "Adding and deleting scripts" on page 285
- "Detecting conflicts with upload_update scripts (deprecated)" on page 106
- "Resolving conflicts with upload_update scripts" on page 109
- "upload_delete table event" on page 436
- "upload_fetch table event" on page 454
- "upload_insert table event" on page 487

SQL example

This example handles updates made to the Customer table in the remote database. The script updates the values in a table named Customer in the consolidated database.

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'upload_update',
'UPDATE Customer
   SET name = {ml r.name}, rep_id = {ml r.rep_id}
      WHERE cust_id = {ml o.cust_id}')
```

This next example performs a similar update, but uses the old (pre-image) values to ensure the update only happens if there is no conflict. If there is a conflict, the update is ignored in this "first in wins" conflict resolution policy.

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'upload_update',
'UPDATE Customer
   SET name = {ml r.name}, rep_id = {ml r.rep_id}
      WHERE cust_id = {ml o.cust_id}')
        AND name = {ml o.name}
        AND rep_id = {ml o.rep_id}
```

MobiLink server APIs

This section describes the MobiLink server APIs for Java and .NET.

Writing synchronization scripts in Java

You control the actions of the MobiLink server by writing synchronization scripts. You can implement these scripts in SQL, .NET or Java. Java synchronization logic can function just as SQL logic functions: the MobiLink server can make calls to Java methods on the occurrence of MobiLink events just as it accesses SQL scripts on the occurrence of MobiLink events. A Java method can return a SQL string to MobiLink.

This section tells you how to set up, develop, and run Java synchronization logic. It includes a sample application and the MobiLink server API for Java Reference.

See also

- "Tutorial: Using Java synchronization logic" [*MobiLink - Getting Started*]
- "Options for writing server-side synchronization logic" [*MobiLink - Getting Started*]
- "Writing synchronization scripts" on page 263

Setting up Java synchronization logic

When you install SQL Anywhere, the installer automatically sets the location of the MobiLink server API for Java classes. When you start the MobiLink server, it automatically includes these classes in your classpath. The MobiLink server API for Java classes are located in *install-dir\Java\mlscript.jar*.

To implement synchronization scripts in Java

1. Create your own class or classes. Write a method for each required synchronization script. These methods must be public. The class must be public in the package.

   See "Methods" on page 527.

   Each class with non-static methods should have a public constructor. The MobiLink server automatically instantiates each class the first time a method in that class is called.

   See "Constructors" on page 527.

2. When compiling the class, you must include the JAR file *java\mlscript.jar*.

   For example,

   ```
   javac MyClass.java -classpath "C:\Program Files\SQL Anywhere 12\java
   \mlscript.jar"
   ```

3. In the MobiLink system tables on your consolidated database, specify the name of the package, class, and method to call for each synchronization script. One class is permitted per script version.

   For example, you can add this information to the MobiLink system tables using the ml_add_java_connection_script stored procedure or the ml_add_java_table_script stored procedure.

   For example, the following SQL statement, when run in a SQL Anywhere database, specifies that for the script version ver1, myPackage.myClass.myMethod should be run whenever the authenticate_user connection-level event occurs. The method that is specified must be the fully qualified name of a public Java method, and the name is case sensitive.

   ```
   call ml_add_java_connection_script('ver1',
   'authenticate_user', 'myPackage.myClass.myMethod')
   ```

   For more information about adding scripts, see:

   - "System procedures to add or delete scripts" on page 691
   - "ml_add_java_connection_script system procedure" on page 698
   - "ml_add_java_table_script system procedure" on page 699

4. Instruct the MobiLink server to load classes. A vital part of setting up Java synchronization logic is to tell the Java VM where to look for Java classes. There are two ways to do this:

   - Use the mlsrv12 -sl java -cp option to specify a set of directories or jar files in which to search for classes. For example, run the following command:

     ```
     mlsrv12 -c "dsn=consolidated1" -sl java (-cp %classpath%;c:\local\Java
     \myclasses.jar)
     ```

     The MobiLink server automatically appends the location of the MobiLink server API for Java classes (java\mlscript.jar) to the set of directories or jar files. The -sl java option also forces the Java VM to load on server startup.

     For more information about the available Java options, see "-sl java mlsrv12 option" on page 65.

   - Explicitly set the classpath. To set the classpath for user-defined classes, use a statement such as the following:

     ```
     SET classpath=%classpath%;c:\local\Java\myclasses.jar
     ```

     If your system classpath includes your Java synchronization logic classes, you do not need to make changes to your MobiLink server command line.

     You can use the -sl java option to force the Java VM to load at server startup. Otherwise, the Java VM is started when the first Java method is executed.

     For more information about the available Java options, see "-sl java mlsrv12 option" on page 65.

5. On Unix, if you want to load a specific JRE, you should set the LD_LIBRARY_PATH (LIBPATH on IBM AIX, SHLIB_PATH on HP-UX) to include the directory containing the JRE. The directory must be listed before any of the SQL Anywhere installation directories.

See also

- "Writing Java synchronization logic" on page 525
- "Java synchronization example" on page 532
- "Tutorial: Using Java synchronization logic" [*MobiLink - Getting Started*]
- "MobiLink server API for Java reference" on page 536
- "Options for writing server-side synchronization logic" [*MobiLink - Getting Started*]
- "Writing synchronization scripts" on page 263

Writing Java synchronization logic

Writing Java synchronization logic requires knowledge of MobiLink events, some knowledge of Java, and knowledge of the MobiLink server API for Java.

For a complete description of the API, see "MobiLink server API for Java reference" on page 536.

Java synchronization logic can be used to maintain state information, and implement logic around the upload and download events. For example, a begin_synchronization script written in Java could store the MobiLink user name in a variable. Scripts called later in the synchronization process can access this variable. Also, you can use Java to access rows in the consolidated database, before or after they are committed.

Using Java reduces dependence on the consolidated database. Behavior is affected less by upgrading the consolidated database to a new version or switching to a different database management system.

Direct row handling

You can use MobiLink direct row handling to communicate remote data to any central data source, application, or web service. Direct row handling uses special classes in the MobiLink server APIs for Java or .NET for direct access to synchronized data.

See "Direct row handling" on page 671.

Class instances

The MobiLink server instantiates your classes at the connection level. When an event is reached for which you have written a non-static Java method, the MobiLink server automatically creates an instance of the class, if it has not already done so on the present connection.

See "Constructors" on page 592.

All methods directly associated with a connection-level or table-level event for one script version **must belong to the same class**.

For each database connection, once a class has been instantiated, the class persists until that connection is closed. So, the same instance may be used for multiple consecutive synchronization sessions. Unless it is explicitly cleared, information present in public or private variables persists across synchronizations that occur on the same connection.

You can also use static classes or variables. In this case, the values are available across all connections.

The MobiLink server automatically deletes your class instances only when the connection to the consolidated database is closed.

Transactions

The normal rules regarding transactions apply to Java methods. The start and duration of database transactions is critical to the synchronization process. Transactions must be started and ended only by the MobiLink server. Explicitly committing or rolling back transactions on the synchronization connection within a Java method violates the integrity of the synchronization process and can cause errors.

These rules apply only to the database connections created by the MobiLink server and, in particular, to SQL statements returned by methods. If your classes create other database connections, use existing management rules to manage classes created by other database connections.

SQL-Java data types

The following table shows SQL data types and the corresponding Java data types.

SQL data type	Corresponding Java data type
VARCHAR	java.lang.String
CHAR	java.lang.String
INTEGER	int or Integer
BINARY	byte[ ]
BIGINT	long
TIMESTAMP	java.sql.Timestamp
INOUT INTEGER	ianywhere.ml.script.InOutInteger
INOUT VARCHAR	ianywhere.ml.script.InOutString
INOUT CHAR	ianywhere.ml.script.InOutString
INOUT BYTEARRAY	ianywhere.ml.script.InOutByteArray
INOUT TIMESTAMP	java.sql.Timestamp

The MobiLink server automatically adds the ianywhere.ml.script package to your classpath if it is not already present. However, when you compile your class you need to add the path of *install-dir\java\mlscript.jar*.

Constructors

The constructor of your class may have one of two possible signatures.

```
public MyScriptClass(ianywhere.ml.script.DBConnectionContext sc)
```

or

```
public MyScriptClass()
```

The synchronization context passed to you is for the connection through which the MobiLink server is synchronizing the current user.

The DBConnectionContext.getConnection method returns the same database connection that MobiLink is using to synchronize the present user. You can execute statements on this connection, but you must not commit or roll back the transaction. The MobiLink server manages the transactions.

The MobiLink server prefers to use constructors with the first signature. It only uses the non-argument constructor if a constructor with the first signature is not present.

See "DBConnectionContext interface" on page 536.

Methods

In general, you implement one method for each synchronization event. These methods must be public. If they are private, the MobiLink server cannot use them and fails to recognize that they exist.

The names of the methods are not important, as long as the names match the names specified in the ml_script table in the consolidated database. In the examples included in the documentation, however, the method names are the same as those of the MobiLink events because this naming convention makes the Java code easier to read.

The signature of your method should match the signature of the script for that event, except that you can truncate the parameter list if you do not need the values of parameters at the end of the list. You should accept only the parameters you need, because overhead is associated with passing the parameters.

You cannot, however, overload the methods. Only one method prototype per class may appear in the ml_script system table.

Registering methods

After creating a method, you must register it. Registering the method creates a reference to the method in the MobiLink system tables on the consolidated database, so that the method is called when the event occurs. You register methods in the same way that you add synchronization scripts, except instead of adding the entire SQL script to the MobiLink system table, you add only the method name.

See "Adding and deleting scripts" on page 285.

Return values

Methods called for a SQL-based upload or download must return a valid SQL language statement. The return type of these methods must be java.lang.String. No other return types are allowed.

The return type of all other scripts must either be java.lang.String or void. No other types are allowed. If the return type is a string and not null, the MobiLink server assumes that the string contains a valid SQL statement and executes this statement in the consolidated database as it would an ordinary SQL-language synchronization script. If a method ordinarily returns a string but does not want to execute a SQL statement against the database upon its return, it can return null.

Debugging Java classes

MobiLink provides various information and facilities that you may find helpful when debugging your Java code. This section describes where you can find this information and how you can exploit these capabilities.

Information in the MobiLink server's log file

The MobiLink server writes messages to a message log file. The server log file contains the following information:

- The Java Runtime Environment. You can use the -jrepath option to request a particular JRE when you start the MobiLink server. The default path is the path of the JRE installed with SQL Anywhere 12.
- The path of the standard Java classes loaded. If you did not specify these explicitly, the MobiLink server automatically adds them to your classpath before invoking the Java VM.
- The fully specified names of the specific methods invoked. You can use this information to verify that the MobiLink server is invoking the correct methods.
- Any output written in a Java method to java.lang.System.out or java.lang.System.err is redirected to the MobiLink server log file.
- The mlsrv12 command line option -verbose can be used.

  See "-v mlsrv12 option" on page 70.

Using a Java debugger

You can debug your Java classes using a standard Java debugger. Specify the necessary parameters using the -sl java option on the mlsrv12 command line.

See "-sl java mlsrv12 option" on page 65.

Specifying a debugger causes the Java VM to pause and wait for a connection from a Java debugger.

Printing information from Java

Alternatively, you may choose to add statements to your Java methods that print information to the MobiLink message log, using java.lang.System.err or java.lang.System.out. Doing so can help you track the progress and behavior of your classes.

Performance tip
Printing information in this manner is a useful monitoring tool, but is not recommended in a production scenario.

The same technique can be exploited to log arbitrary synchronization information or collect statistical information on how your scripts are used.

Writing your own test driver

You may want to write your own driver to exercise your Java classes. This approach can be helpful because it isolates the actions of your Java methods from the rest of the MobiLink system.

Handling MobiLink server errors in Java

When scanning the log is not enough, you can monitor your applications programmatically. For example, you can send messages of a certain type in an email.

You can write methods that are passed a class representing every error or warning message that is printed to the log. This may help you monitor and audit a MobiLink server.

The following code installs a LogListener for all warning messages, and writes the information to a file.

```
class TestLogListener implements LogListener {
    FileOutputStream _out_file;

    public TestLogListener(FileOutputStream out_file) {
        _out_file = out_file;
    }

    public void messageLogged(ServerContext sc, LogMessage msg) {
        String  type;
        String  user;

        try {
            if (msg.getType() == LogMessage.ERROR) {
                type = "ERROR";
            } else if (msg.getType() == LogMessage.WARNING) {
                type = "WARNING";
            }
            else {
                type = "UNKNOWN!!!";
            }

            user = msg.getUser();
            if (user == null) {
                user = "NULL";
            }
            _out_file.write(("Caught msg type="
                + type
                + " user=" + user
                + " text=" + msg.getText()
                + "\n").getBytes()
            );
            _out_file.flush();
        }
        catch(Exception e) {
```

```
                // Print some error output to the MobiLink log.
                e.printStackTrace();
            }
        }
    }
```

The following code registers TestLogListener to receive warning messages. Call this code from anywhere that has access to the ServerContext such as a class constructor or synchronization script.

```
// ServerContext serv_context;
serv_context.addWarningListener(
      new MyLogListener(ll_out_file)
);
```

See also

- "addErrorListener method" on page 562
- "removeErrorListener method" on page 566
- "addWarningListener method" on page 562
- "removeWarningListener method" on page 567
- "LogListener interface" on page 554
- "LogMessage class" on page 555

User-defined start classes

You can define start classes that are loaded automatically when the server is started. The purpose of this feature is to allow you to write Java code that executes at the time the MobiLink server starts the Java VM —before the first synchronization. This means you can create connections or cache data before a user synchronization request.

You do this with the DMLStartClasses option of the mlsrv12 -sl java option. For example, the following is part of a mlsrv12 command line. It causes mycl1 and mycl2 to be loaded as start classes.

```
-sl java(-DMLStartClasses=com.test.mycl1,com.test.mycl2)
```

Classes are loaded in the order in which they are listed. If the same class is listed more than once, more than one instance is created.

All start classes must be public and must have a public constructor that either accepts no arguments or accepts one argument of type ianywhere.ml.script.ServerContext.

The names of loaded start classes are output to the MobiLink log with the message "Loaded JAVA start class: *classname*".

For more information about Java VM options, see "-sl java mlsrv12 option" on page 65.

To see the start classes that are constructed at server start time, see "getStartClassInstances method" on page 565.

Example

The following is a start class template. It starts a daemon thread that processes events and creates a database connection. (Not all start classes need to create a thread but if a thread is spawned it should be a daemon thread.)

```
import ianywhere.ml.script.*;
import java.sql.*;

public class StartTemplate extends
    Thread implements ShutdownListener {
    ServerContext   _sc;
    Connection      _conn;
    boolean         _exit_loop;

    public StartTemplate(ServerContext sc) throws SQLException {

        // Perform setup first so that an exception
        // causes MobiLink startup to fail.
        _sc = sc;

        // Create a connection for use later.
        _conn = _sc.makeConnection();

        _exit_loop = false;
        setDaemon(true);
        start();
    }

    public void run() {
        _sc.addShutdownListener(this);

        // run() cannot throw exceptions.
        try {
            handlerLoop();
            _conn.close();
            _conn = null;
        }
        catch(Exception e) {

            // Print some error output to the MobiLink log.
            e.printStackTrace();

            // This thread shuts down and so does not
            // need to be notified of shutdown.
            _sc.removeShutdownListener(this);

            // Ask server to shutdown so that this fatal
            // error is fixed.
            _sc.shutdown();
        }
        // Shortly after return this thread no longer exists.
        return;
    }

    // stop our event handler loop
    public void shutdownPerformed(ServerContext sc) {
        try {
            // Wait max 10 seconds for thread to die.
            join(10*1000);
        }
        catch(Exception e) {
            // Print some error output to the MobiLink log.
```

```
                e.printStackTrace();
            }
        }

        private void handlerLoop() throws InterruptedException {
            while (!_exit_loop) {
                // Handle events in this loop. Sleep not
                // needed, block on event queue.
                sleep(1 * 1000);
            }
        }
    }
```

Java synchronization example

Java synchronization logic works with MobiLink and common Java classes to provide you with flexibility in deploying applications using MobiLink server. The following section introduces you to this extended range of functionality using a simple example.

This section describes a working example of Java synchronization logic. Before you try to use this class or write your own class, use the following checklist to ensure you have all the pieces in place before compiling the class.

- Plan your functionality using, for example, pseudocode.
- Create a map of database tables and columns.
- Configure the consolidated database for Java synchronization by ensuring you have specified in the MobiLink system tables the language type and location of the Java synchronization methods.

  See "Setting up Java synchronization logic" on page 523.
- Create a list of associated Java classes that are called during the running of your Java class.
- Store your Java classes in a location that is in the classpath for MobiLink server.

Plan

The Java synchronization logic for this example points to the associated Java files and classes that contain functionality needed for the example to work. It shows you how to create a class CustEmpScripts. It shows you how to set up a synchronization context for the connection. Finally, the example provides Java methods to

- authenticate a MobiLink user.
- perform download and upload operations using cursors for each database table.

Schema

The tables to be synchronized are emp and cust. The emp table has three columns called emp_id, emp_name and manager. The cust table has three columns called cust_id, cust_name and emp_id. All columns in each table are synchronized. The mapping from consolidated to remote database is such that

the table names and column names are identical in both databases. One additional table, an audit table, is added to the consolidated database.

Java class files

The files used in the example are included in the *Samples\MobiLink\JavaAuthentication* directory.

Setup

The following code sets up the Java synchronization logic. The import statements tell the Java VM the location of needed files. The public class statement declares the class.

```
// Use a package when you create your own script.
import ianywhere.ml.script.InOutInteger;
import ianywhere.ml.script.DBConnectionContext;
import ianywhere.ml.script.ServerContext;
import java.sql.*;

public class CustEmpScripts {
    // Context for this synchronization connection.
    DBConnectionContext _conn_context;

    // Same connection MobiLink uses for sync.
    // Do not commit or close this.
    Connection _sync_connection;
    Connection _audit_connection;

    //Get a user id given the user name. On audit connection.
    PreparedStatement _get_user_id_pstmt;

    // Add record of user logins added. On audit connection.
    PreparedStatement _insert_login_pstmt;

    // Prepared statement to add a record to the audit table.
    // On audit connection.
    PreparedStatement _insert_audit_pstmt;

    // ...
}
```

The CustEmpScripts constructor sets up all the prepared statements for the authenticateUser method. It sets up member data.

```
public CustEmpScripts(DBConnectionContext cc) throws SQLException {
    try {
        _conn_context = cc;
        _sync_connection = _conn_context.getConnection();

        ServerContext serv_context =
        _conn_context.getServerContext();
        _audit_connection = serv_context.makeConnection();

        // Get the prepared statements ready.
        _get_user_id_pstmt =
            _audit_connection.prepareStatement(
                "select user_id from ml_user where name = ?"
            );

        _insert_login_pstmt =
            _audit_connection.prepareStatement(
                "INSERT INTO login_added(ml_user, add_time) "
                    + "VALUES (?, { fn CONVERT({ fn NOW() }, SQL_VARCHAR) })"
```

```
            );

        _insert_audit_pstmt =
            _audit_connection.prepareStatement(
               "INSERT INTO login_audit(ml_user_id, audit_time, audit_action)
"
                    + "VALUES (?, { fn CONVERT({ fn NOW() },
SQL_VARCHAR) }, ?)"
            );
    }
    catch(SQLException e) {
        freeJDBCResources();
        throw e;
    }
    catch(Error e) {
        freeJDBCResources();
        throw e;
    }
}
```

The finalize method cleans up JDBC resources if end_connection is not called. It calls the freeJDBCResources method, which frees allocated memory and closes the audit connection.

```
protected void finalize() throws SQLException, Throwable {
    super.finalize();
    freeJDBCResources();
}

private void freeJDBCResources() throws SQLException {
    if (_get_user_id_pstmt != null) {
        _get_user_id_pstmt.close();
    }
    if (_insert_login_pstmt != null) {
        _insert_login_pstmt.close();
    }
    if (_insert_audit_pstmt != null) {
        _insert_audit_pstmt.close();
    }
    if (_audit_connection != null) {
        _audit_connection.close();
    }
    _conn_context       = null;
    _sync_connection    = null;
    _audit_connection   = null;
    _get_user_id_pstmt  = null;
    _insert_login_pstmt = null;
    _insert_audit_pstmt = null;
}
```

The endConnection method cleans up resources once the resources are not needed.

```
public void endConnection() throws SQLException {
    freeJDBCResources();
}
```

The authenticateUser method below approves all user logins and logs user information to database tables. If the user is not in the ml_user table they are logged to login_added. If the user id is found in ml_user then they are logged to login_audit. In a real system you would not ignore the user_password, but this sample approves all users for simplicity. The endConnection method throws SQLException if any of the database operations fail with an exception.


```
public void authenticateUser(
    InOutInteger authentication_status,
    String user_name) throws SQLException
{

    boolean new_user;
    int user_id;

    // Get ml_user id.
    _get_user_id_pstmt.setString(1, user_name);

    ResultSet user_id_rs =
    _get_user_id_pstmt.executeQuery();
    new_user = !user_id_rs.next();
    if (!new_user) {
        user_id = user_id_rs.getInt(1);
    }
    else {
        user_id = 0;
    }

    user_id_rs.close();
    user_id_rs = null;

    // In this tutorial always allow the login.
    authentication_status.setValue(1000);

    if (new_user) {
        _insert_login_pstmt.setString(1, user_name);
        _insert_login_pstmt.executeUpdate();
        java.lang.System.out.println("user: " + user_name + " added. ");
    }
    else {
        _insert_audit_pstmt.setInt(1, user_id);
        _insert_audit_pstmt.setString(2, "LOGIN ALLOWED");
        _insert_audit_pstmt.executeUpdate();
    }
    _audit_connection.commit();
    return;
}
```

The following methods use SQL statements to act as cursors on the database tables. Since these are cursor scripts, they must return a SQL string.

```
public static String empUploadInsertStmt() {
    return("INSERT INTO emp(emp_id, emp_name) VALUES(?, ?)");
}

public static String empUploadDeleteStmt() {
    return("DELETE FROM emp WHERE emp_id = ?");
}

public static String empUploadUpdateStmt() {
    return("UPDATE emp SET emp_name = ? WHERE emp_id = ?");
}

public static String empDownloadCursor() {
    return("SELECT emp_id, emp_name FROM emp");
}

public static String custUploadInsertStmt() {
    return("INSERT INTO cust(cust_id, emp_id, cust_name) VALUES (?, ?, ?)");
}
```

```
public static String custUploadDeleteStmt() {
    return("DELETE FROM cust WHERE cust_id = ?");
}

public static String custUploadUpdateStmt() {
    return("UPDATE cust SET emp_id = ?, cust_name = ? WHERE cust_id = ?");
}

public static String custDownloadCursor() {
    return("SELECT cust_id, emp_id, cust_name FROM cust");
}
```

Use the following command to compile the code:

```
javac -cp %sqlany12%\java\mlscript.jar CustEmpScripts.java
```

Run the MobiLink server with the location of CustEmpScripts.class in the classpath. The following is a partial command line:

```
mlsrv12 ... -sl java (-cp <class_location>)
```

MobiLink server API for Java reference

This section explains the MobiLink Java interfaces and classes, and their associated methods and constructors. To use these classes, reference the *mlscript.jar* assembly, located in *install-dir\java*.

DBConnectionContext interface

Syntax

public **ianywhere.ml.script.DBConnectionContext**

Remarks

Interface for obtaining and accessing information about the current database connection. A DBConnectionContext instance is passed to the constructor of classes containing scripts. If context is required for a background thread or beyond the lifetime of a connection, use the ServerContext class.

See also

- “Constructors” on page 527
- “ServerContext interface” on page 560

Members

All members of **ianywhere.ml.script.DBConnectionContext**, including all inherited members.

Example

The following example shows you how to create a class level DBConnectionContext instance to use in your synchronization scripts. The DBConnectionContext getConnection method obtains a java.sql.Connection instance representing the current connection with the MobiLink consolidated database.

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

public class OrderProcessor {
    DBConnectionContext _cc;

    public OrderProcessor(DBConnectionContext cc) {
        _cc = cc;
    }

    // The method used for the handle_DownloadData event.
    public void HandleEvent() throws SQLException {
        java.sql.Connection my_connection = _cc.getConnection();
        // ...
    }

    // ...
}
```

Caution
A DBConnectionContext instance should not be used outside the thread that calls into your Java code.

getConnection method

Syntax

public java.sql.Connection **getConnection( )**
throws java.sql.SQLException

Remarks

Returns the existing connection with the MobiLink consolidated database as a JDBC connection. This connection is the same connection that MobiLink uses when executing SQL scripts for this synchronization.

This connection must not be committed, closed or altered in any way that would affect the MobiLink server use of this connection. The connection returned is only valid for the lifetime of the underlying

MobiLink connection. Do not use the connection after the end_connection event has been called for that connection.

If an error occurs binding the existing connection as a JDBC connection then it throws java.sql.SQLException

If a server connection with full access is required, use ServerContext.makeConnection().

Returns

The existing connection with the MobiLink consolidated database as a JDBC connection.

Example

See "DBConnectionContext interface" on page 536.

getDownloadData method

Syntax

public DownloadData **getDownloadData()**

Remarks

Returns a DownloadData instance for the current synchronization. Use the DownloadData class to create the download for direct row handling.

Returns

A DownloadData instance for the current synchronization.

See also

- "DownloadData interface" on page 542
- "Direct row handling" on page 671

Example

The following example shows you how to obtain a DownloadData instance for the current synchronization using the DBConnectionContext getDownloadData method.

Note
This example assumes you have a DBConnectionContext instance called _cc.

```
// The method used for the handle_DownloadData event.
public void HandleDownload() throws SQLException {
    // get the DownloadData for the current synchronization
    DownloadData my_dd = _cc.getDownloadData();

    // ...
}

// ...
```


getNamedParameter method

Returns the value of the named parameter.

Syntax

public String **getNamedParameter(**
String *parameter_name*
)

Returns

Returns the value of the named parameter, may be null

getProperties method

Syntax

public java.util.Properties **getProperties( )**

Remarks

Returns the properties for this connection, based on this connection's script version. Properties are stored in the ml_property table.

Consult your Java Software Development Kit documentation for more information about java.util.Properties.

Returns

The properties for this connection.

See also

- "ml_add_property system procedure" on page 705

Example

The following example shows you how to output the properties for a DBConnectionContext.

Note
This example assumes you have a DBConnectionContext instance called _cc.

```
// The method used to output the connection properties.
public void outputProperties() {
    // output the Properties for the current synchronization
    java.util.Properties properties = _cc.getProperties();
    System.out.println(properties.toString());
}
```

getRemoteID method

Syntax

public java.lang.String **getRemoteID( )**

Remarks

Returns the remote ID of the database currently synchronizing on this connection.

Returns

The remote ID.

See also

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

Example

The following example shows you how to output the remote ID for a DBConnectionContext.

Note
This example assumes you have a DBConnectionContext instance called _cc.

```
// The method used to output the remote ID.
public void outputRemoteID() {
    // output the Remote ID for the current synchronization
    String remoteID = _cc.getRemoteID();
    System.out.println(remoteID);
}
```

getServerContext method

Syntax

public ServerContext **getServerContext( )**

Remarks

Returns the ServerContext for this MobiLink server.

Returns

The ServerContext for this MobiLink server.

See also

- "ServerContext interface" on page 560

Example

The following example shows you how to get the ServerContext instance for a DBConnectionContext and shut down the server.

Note
This example assumes you have a DBConnectionContext instance called _cc.


```
// A method that uses an instance of the ServerContext to shut down the
server
public void shutDownServer() {
    ServerContext context = _cc.getServerContext();
    context.shutdown();
}
```

getVersion method

Syntax

public java.lang.String **getVersion( )**

Remarks

Returns the script version string.

Returns

The script version string.

See also

- "ml_add_property system procedure" on page 705

Example

The following example shows you how to get the script version and use it to make decisions.

> **Note**
> This example assumes you have a DBConnectionContext instance called _cc.

```
// A method that uses the script version
public void handleEvent() {
    // ...

    String version = _cc.getVersion();
    if (version.equals("My Version 1")) {
        // ...
    } else if (version.equals("My Version 2")) {
        // ...
    }
}

// ...
```

setNamedParameter method

Sets the named parameter.

Syntax

public void **getVersion()**
String *parameter_name*,

String *value*
)

Remarks

The value must be less than 128 bytes.

DownloadData interface

Syntax

public **ianywhere.ml.script.DownloadData**

Remarks

Encapsulates download data operations for direct row handling. To obtain a DownloadData instance, use the DBConnectionContext getDownloadData method.

Use the DownloadData.getDownloadTables and getDownloadTableByName methods to return DownloadTableData instances.

This download data is available through DBConnectionContext. It is not valid to access the download data before the begin_synchronization event or after the end_download event. It is not valid to access DownloadData in an upload-only synchronization.

See also

- "DownloadTableData interface" on page 544
- "handle_DownloadData connection event" on page 383
- DBConnectionContext "getDownloadData method" on page 538
- "Direct row handling" on page 671

Members

All members of **ianywhere.ml.script.DownloadData**, including all inherited members.

- "getDownloadTableByName method" on page 543
- "getDownloadTables method" on page 544

Example

The following example shows you how to obtain a DownloadData instance for the current synchronization using the DBConnectionContext getDownloadData method.

```
DBConnectionContext _cc;

// Your class constructor.
public OrderProcessor(DBConnectionContext cc) {
    _cc = cc;
}

// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();
```



```
    // ...
}
```

getDownloadTableByName method

Syntax

public DownloadTableData getDownloadTableByName(
string *table-name*);

Remarks

Gets the named download table for this synchronization. Returns null if there is no table with the given name in this synchronization.

Parameters

- **table_name** The name of the table for which you want the download data.

Returns

A DownloadTableData instance representing the specified table, or null if a table of the given name does not exist for the current synchronization.

See also

- "DownloadData interface" on page 542
- "DownloadTableData interface" on page 544
- "DBConnectionContext interface" on page 536
- "Direct row handling" on page 671

Example

The following example uses the getDownloadTableByName method to return a DownloadTableData instance for the remoteOrders table.

> **Note**
> This example assumes you have a DBConnectionContext instance called _cc.

```
// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData my_download_table =
my_dd.getDownloadTableByName("remoteOrders");

    // ...
}
```

getDownloadTables method

Syntax

public DownloadTableData[ ] **getDownloadTables( )**

Remarks

Gets an array of all the tables for download data in this synchronization. The operations performed on this table are sent to the remote database.

Returns

An array of DownloadTableData objects for the current synchronization. The order of tables in the array is the same as the upload order of the remote.

See also

- "DownloadData interface" on page 542
- "DownloadTableData interface" on page 544
- "DBConnectionContext interface" on page 536
- "Direct row handling" on page 671

Example

The following example uses the DownloadData.getDownloadTables method to obtain an array of DownloadTableData objects for the current synchronization. The example assumes you have a DBConnectionContext instance called _cc.

```
// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // Get an array of tables to set download operations.
    DownloadTableData[] download_tables = my_dd.getDownloadTables();

    // Get the first table in the DownloadTableData array.
    DownloadTableData my_download_table = download_tables[0];

    // ...
}
```

DownloadTableData interface

Syntax

public **ianywhere.ml.script.DownloadTableData**

Remarks

Encapsulates table operations for MobiLink direct downloads. Use this interface to set the data operations that are downloaded to the client. To obtain DownloadTableData instances for the current synchronization, use the DownloadData interface. You can use the

DownloadTableData.getUpsertPreparedStatement and getDeletePreparedStatement methods to obtain Java prepared statements for insert and update, and delete operations, respectively. The java.sql.PreparedStatement.executeUpdate method registers an operation for download.

Note
You must set all column values for insert and update prepared statements. For delete operations you set primary key values.

You cannot have both the delete and upsert prepared statements open at the same time.

Consult your Java Software Development Kit documentation for more information about java.sql.PreparedStatement.

Execute the delete statement with all primary keys set to null to have the remote truncate the table.

See also

- "DownloadData interface" on page 542
- "handle_DownloadData connection event" on page 383
- "Direct row handling" on page 671

Members

All members of **ianywhere.ml.script.DownloadTableData**, including all inherited members.

- "getDeletePreparedStatement method" on page 546
- "getUpsertPreparedStatement method" on page 547
- "getName method" on page 549
- "getMetaData method" on page 549
- "getLastDownloadTime method" on page 550

Example

Assume you use a table called remoteOrders in MobiLink client databases.

```
CREATE TABLE remoteOrders (
    pk INT NOT NULL,
    col1 VARCHAR(200),
    PRIMARY KEY (pk)
);
```

The following example uses the DownloadData.getDownloadTableByName method to return a DownloadTableData instance representing the remoteOrders table.

Note
This example assumes you have a DBConnectionContext instance called _cc.

```
// The method used for the handle_DownloadData event
public void handleDownload() throws SQLException {

    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();
```

```
    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData td = my_dd.getDownloadTableByName("remoteOrders");

    // User defined-methods to set download operations.
    setDownloadInserts(td);
    setDownloadDeletes(td);

    // ...
}
```

In this example, the setDownloadInserts method uses the DownloadTableData.getUpsertPreparedStatement to obtain a prepared statement for rows you want to insert or update. The PreparedStatement.setInt and PreparedStatement.setString methods set the column values you want to insert into the remote database.

```
void setDownloadInserts(DownloadTableData td) {
    java.sql.PreparedStatement insert_ps = td.getUpsertPreparedStatement();

    // The following method calls are the same as the following SQL
statement:
    // INSERT INTO remoteOrders(pk, col1) values(2300, "truck");
    insert_ps.setInt(1, 2300);
    insert_ps.setString(2, "truck");

    int update_result = insert_ps.executeUpdate();
    if (update_result == 0) {
        // Insert was filtered because it was uploaded
        // in the same synchronization.
    }
    else {
        // Insert was not filtered.
    }
}
```

The setDownloadDeletes method uses the DownloadTableData.getDeletePreparedStatement to obtain a prepared statement for rows you want to delete. The java.sql.PreparedStatement.setInt method sets the primary key values for rows you want to delete in the remote database and the java.sql.PreparedStatement.executeUpdate method registers the row values for download.

```
void setDownloadDeletes(DownloadTableData td) {
    java.sql.PreparedStatement delete_ps = td.getDeletePreparedStatement();

    // The following method calls are the same as the following SQL
statement:
    // DELETE FROM remoteOrders  where pk=2300;
    delete_ps.setInt(1, 2300);
    delete_ps.executeUpdate();
}
```

getDeletePreparedStatement method

Syntax

public java.sql.PreparedStatement **getDeletePreparedStatement( )** throws **SQLException**

Remarks

Returns a java.sql.PreparedStatement instance that allows the user to add delete operations to the download. The prepared statement applies to the download table and contains a parameter for each primary key column in the table.

The prepared statement applies to the DownloadTableData instance and contains a parameter for each primary key column in the table.

To include a delete operation in the download, set all columns in your java.sql.PreparedStatement and then call the java.sql.PreparedStatement.executeUpdate method.

Set all the parameters to null to have the remote database truncate the table.

> **Note**
> You must set all primary key values for download delete operations, or set all primary key values to null for truncate operations.

Returns

A java.sql.PreparedStatement instance for adding delete operations to the download.

Exceptions

- **SQLException** Thrown if there is a problem retrieving the delete java.sql.PreparedStatement instance.

See also

- "DownloadTableData interface" on page 544
- "Direct row handling" on page 671

Example

In the following example, the setDownloadDeletes method uses the DownloadTableData.getDeletePreparedStatement to obtain a prepared statement for rows you want to delete. The java.sql.PreparedStatement.setInt method sets the primary key values for rows you want to delete in the remote database and the java.sql.PreparedStatement.executeUpdate method sets the row values in the download.

```
void setDownloadDeletes(DownloadTableData td) {
    java.sql.PreparedStatement delete_ps = td.getDeletePreparedStatement();
    // This is the same as executing the following SQL statement:
    // DELETE FROM remoteOrders  where pk=2300;
    delete_ps.setInt(1, 2300);
    delete_ps.executeUpdate();
    delete_ps.close();
 }
```

getUpsertPreparedStatement method

Syntax

public java.sql.PreparedStatement **getUpsertPreparedStatement( )** throws **SQLException**

Remarks

Returns a java.sql.PreparedStatement instance which allows the user to add upsert (insert or update) operations to the download of a synchronization. The prepared statement applies to the DownloadTableData instance and contains a parameter for each column in the table.

To include an insert or update operation in the download, set all column values in your java.sql.PreparedStatement and then call the java.sql.PreparedStatement.executeUpdate method. Calling java.sql.PreparedStatement.executeUpdate on the prepared statement returns 0 if the insert or update operation was filtered and returns 1 if the operation was not filtered. An operation is filtered if it was uploaded in the same synchronization.

Note
You must set all column values for download insert and update operations.

Returns

A java.sql.PreparedStatement instance for adding upsert operations to the download.

Exceptions

- **SQLException** Thrown if there is a problem retrieving the upsert java.sql.PreparedStatement instance.

See also

- "DownloadTableData interface" on page 544
- "Direct row handling" on page 671

Example

In the following example, the setDownloadInserts method uses the DownloadTableData.getUpsertPreparedStatement to obtain a prepared statement for rows you want to insert or update. The java.sql.PreparedStatement.setInt and PreparedStatement.setString methods set the column values, and the PreparedStatement.executeUpdate method sets the row values in the download.

```
void setDownloadInserts(DownloadTableData td) {
    java.sql.PreparedStatement insert_ps = td.getUpsertPreparedStatement();

    // This is the same as executing the following SQL statement:
    // INSERT INTO remoteOrders(pk, col1) VALUES (2300, "truck");
    insert_ps.setInt(1, 2300);
    insert_ps.setString(2, "truck");

    int update_result = insert_ps.executeUpdate();
    if (update_result == 0) {
        // Insert was filtered because it was uploaded
        // in the same synchronization.
    }
    else {
        // Insert was not filtered.
    }
```



```
        insert_ps.close();
    }
```

getName method

Syntax

public java.lang.String **getName( )**

Remarks

Returns the table name for the DownloadTableData instance. You can also access the table name using the java.sql.ResultSetMetaData instance returned by the DownloadTableData.getMetaData method.

Returns

The table name for the DownloadTableData instance.

See also

- "DownloadTableData interface" on page 544
- DownloadTableData "getMetaData method" on page 549
- "Direct row handling" on page 671

Example

The following example shows you how to output the table name for the DownloadTableData instance.

> **Note**
> This example assumes you have a DBConnectionContext instance called _cc.

```
// The method used for the handle_DownloadData event
public void handleDownload() throws SQLException {

    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData td = my_dd.getDownloadTableByName("remoteOrders");

    // Print the table name to standard output (remoteOrders)
    System.out.println(td.getName());

    // User defined-methods to set download operations.
    setDownloadInserts(td);
    setDownloadDeletes(td);

    // ...
}
```

getMetaData method

Syntax

public java.sql.ResultSetMetaData **getMetaData( )**

Remarks

Gets the metadata for the DownloadTableData instance. The metadata is a standard java.sql.ResultSetMetaData object.

If you want the metadata to contain column name information, specify in your client that column names should be sent with the upload.

Consult your Java Software Development Kit documentation for more information about java.sql.ResultSetMetaData.

Returns

The metadata for the DownloadTableData instance.

See also

- "DownloadTableData interface" on page 544
- "Direct row handling" on page 671
- SQL Anywhere clients: "SendColumnNames (scn) extended option" [*MobiLink - Client Administration*]
- UltraLite: "Send Column Names synchronization parameter" [*UltraLite - Database Management and Reference*]

Example

The following example shows you how get the number of columns used in the query for the DownloadTableData instance.

```
import java.sql.ResultSetMetaData;

// The method used to return the number of columns in a DownloadTableData
instance query
public int getNumColumns(DownloadTableData td) {
    ResultSetMetaData rsmd = td.getMetaData();
    return rsmd.getColumnCount();
}
```

getLastDownloadTime method

Syntax

public java.sql.Timestamp **getLastDownloadTime( )**

Remarks

Returns last download time for this table. This is the same last download time passed to several of the per table download events.

The last download time is useful for generating the table download data for a particular synchronization.

Returns

The last download time for this download table.

See also

- "DownloadTableData interface" on page 544
- "Direct row handling" on page 671

Example

The following example is a snippet of code that populates a table in the download with inserts using the last download time. Note that this example assumes you have a DBConnectionContext instance called _cc.

```
// The method used for the handle_DownloadData event
public void handleDownload() throws SQLException {

    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData td = my_dd.getDownloadTableByName("remoteOrders");

    // get the inserts given a last download time
    ResultSet inserts_rs  =
makeInsertsFromTimestamp(td.getLastDownloadTime());

    // fill the DownloadTableData using the inserts resultset.
    setDownloadInsertsFromRS(td, inserts_rs);
    inserts_rs.close();

    // ...
}
```

InOutInteger interface

Syntax

public **ianywhere.ml.script.InOutInteger**

Remarks

Passed into methods to enable the functionality of an in/out parameter passed to a SQL script.

Members

All members of **ianywhere.ml.script.InOutInteger**, including all inherited members.

- "getValue method" on page 552
- "setValue method" on page 552

Example

The following call to a MobiLink system procedure registers a Java method called handleError as the script for the handle_error connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
    'ver1',
    'handle_error',
    'ExamplePackage.ExampleClass.handleError'
)
```

The following is the sample Java method handleError. It processes an error based on the data that is passed in. It also determines the resulting error code.

```
public String handleError(
    ianywhere.ml.script.InOutInteger actionCode,
    int errorCode,
    String errorMessage,
    String user,
    String table)
{

    int new_ac;
    if (user == null) {
        new_ac = handleNonSyncError(errorCode, errorMessage);
    } else if (table == null) {
        new_ac = handleConnectionError(errorCode, errorMessage, user);
    }
    else {
        new_ac = handleTableError(errorCode, errorMessage, user, table);
    }

    // Keep the most serious action code.
    if (actionCode.getValue() < new_ac) {
        actionCode.setValue(new_ac);
    }
}
```

getValue method

Syntax

public int **getValue( )**

Remarks

Returns the value of this integer parameter.

Returns

The value of this integer parameter.

Example

See: "InOutInteger interface" on page 551.

setValue method

Syntax

public void **setValue(** int *new_value* **)**

Remarks

Sets the value of this integer parameter.

Parameters

- **new_value** The value for this integer to take.

Example

See: "InOutInteger interface" on page 551.

InOutString interface

Syntax

public **ianywhere.ml.script.InOutString**

Remarks

Passed into methods to enable the functionality of an in/out parameter passed to a SQL script.

Members

All members of **ianywhere.ml.script.InOutString**, including all inherited members.

- "getValue method" on page 554
- "setValue method" on page 554

Example

The following call to a MobiLink system procedure registers a Java method called modifyUser as the script for the modify_user connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
    'ver1',
    'modify_user',
    'ExamplePackage.ExampleClass.modifyUser'
)
```

The following is the sample Java method modifyUser. It gets the user ID from the database and then uses it to set the user name.

```
public String modifyUser(InOutString io_user_name) throws SQLException {
    Statement uid_select = curConn.createStatement();
    ResultSet uid_result = uid_select.executeQuery(
        "SELECT rep_id FROM SalesRep WHERE name = '"
            + io_user_name.getValue() + "' "
    );
    uid_result.next();
    io_user_name.setValue(java.lang.Integer.toString(uid_result.getInt(1));
    uid_result.close();
    uid_select.close();
    return (null);
}
```

getValue method

Syntax

public java.lang.String **getValue( )**

Remarks

Returns the value of this string parameter.

Returns

The value of this string parameter.

Example

See: "InOutString interface" on page 553

setValue method

Syntax

public void **setValue(** java.lang.String *new_value* **)**

Remarks

Sets the value of this String parameter.

Parameters

- **new_value** The value for this String to take.

Example

See: "InOutString interface" on page 553

LogListener interface

Syntax

public **ianywhere.ml.script.LogListener**

Remarks

The listener interface for catching messages that are printed to the log.

See also

- "Handling MobiLink server errors in Java" on page 529

Members

All members of **ianywhere.ml.script.LogListener**, including all inherited members.

- "messageLogged method" on page 555

Example

See: "Handling MobiLink server errors in Java" on page 529

messageLogged method

Syntax

public void **messageLogged(**
ServerContext *sc*,
LogMessage *message* **)**

Remarks

Invoked when a message is printed to the log.

Parameters

- **sc** The context for the server that is printing the message.
- **message** The LogMessage that has been sent to the MobiLink log.

Example

See: "Handling MobiLink server errors in Java" on page 529

LogMessage class

Syntax

public **ianywhere.ml.script.LogMessage**

Remarks

Holds the data associated with a log message.

Extends java.lang.Object.

See also

- "Handling MobiLink server errors in Java" on page 529

Members

All members of **ianywhere.ml.script.LogMessage**, including all inherited members.

Example

The following example installs a LogListener for all warning, error, and info messages, then writes the information to a file.

The following code installs a LogListener for all warning messages.

```
class WarningLogListener implements LogListener {
    FileOutputStream _outFile;

    public WarningLogListener(FileOutputStream outFile) {
        _outFile = outFile;
    }

    public void messageLogged(ServerContext sc, LogMessage msg) {
        String user;

        try {
            if (msg.getType() != LogMessage.WARNING) {
                //this class deals exclusively with warnings
                return;
            }
            user = msg.getUser();

            if (user == null) {
                user = "NULL";
            }

            _outFile.write(("Caught warning"
                + " user=" + user
                + " text=" + msg.getText()
                + "\n").getBytes()
            );
            _outFile.flush();
        }
        catch(Exception e) {
            // Print some error output to the MobiLink log.
            e.printStackTrace();
        }
    }
}
```

The following code installs a LogListener for all error messages.

```
class ErrorLogListener implements LogListener {
    FileOutputStream _outFile;

    public ErrorLogListener(FileOutputStream outFile) {
        _outFile = outFile;
```



```
    }

    public void messageLogged(ServerContext sc, LogMessage msg) {
        String user;

        try {
            if (msg.getType() != LogMessage.ERROR) {
                //this class deals exclusively with errors
                return;
            }

            user = msg.getUser();
            if (user == null) {
                user = "NULL";
            }

            _outFile.write(("Caught error"
                    + " user=" + user
                    + " text=" + msg.getText()
                    + "\n").getBytes()
            );
            _outFile.flush();
        }
        catch(Exception e) {
            // Print some error output to the MobiLink log.
            e.printStackTrace();
        }
    }
}
```

The following code installs a LogListener for all info messages.

```
class InfoLogListener implements LogListener {
    FileOutputStream _outFile;

    public InfoLogListener(FileOutputStream outFile) {
        _outFile = outFile;
    }

    public void messageLogged(ServerContext sc, LogMessage msg) {
        String user;

        try {
            if (msg.getType() != LogMessage.INFO) {
                // this class deals exclusively with info
                return;
            }

            user = msg.getUser();
            if (user == null) {
                user = "NULL";
            }

            _outFile.write(("Caught info"
                    + " user=" + user
                    + " text=" + msg.getText()
                    + "\n").getBytes()
            );
            _outFile.flush();
        }
        catch(Exception e) {
            // Print some error output to the MobiLink log.
            e.printStackTrace();
```

```
            }
        }
    }
```

The following code registers WarningLogListener, ErrorLogListener, and InfoLogListener to receive warning, error, and info messages respectively. Call this code from anywhere that has access to the ServerContext such as a class constructor or synchronization script.

```
// ServerContext serv_context;
// FileOutputStream outFile
serv_context.addWarningListener(new WarningLogListener(outFile));
serv_context.addErrorListener(new ErrorLogListener(outFile));
serv_context.addInfoListener(new InfoLogListener(outFile));
```

ERROR variable

Syntax

int **ERROR**

Remarks

Indicates that the log message is an error.

Example

See: “LogMessage class” on page 555.

INFO variable

Syntax

int **INFO**

Remarks

Indicates that the message log displays information.

Example

See: “addInfoListener method” on page 560.

WARNING variable

Syntax

int **WARNING**

Remarks

Indicates that the log message is a warning.

Example

See: "LogMessage class" on page 555.

getType method

Syntax

public int **getType( )**

Remarks

Accessor for this message type.

Returns

The type of this message, which can be either LogMessage.ERROR, LogMessage.INFO, or LogMessage.WARNING.

Example

See: "LogMessage class" on page 555.

getUser method

Syntax

public java.lang.String **getUser( )**

Remarks

Accessor for this message user. If the message has no user, then the user is null.

Returns

The user associated with this message.

Example

See: "LogMessage class" on page 555.

getText method

Syntax

public java.lang.String **getText( )**

Remarks

Accessor for the message text.

Returns

The main text of this message.

Example

See: “LogMessage class” on page 555.

ServerContext interface

Syntax

public **ianywhere.ml.script.ServerContext**

Remarks

An instantiation of all the context that is present for the duration of the MobiLink server. This context can be held as static data and used in a background thread. It is valid for the duration of the Java VM invoked by MobiLink.

To access a ServerContext instance, use the DBConnectionContext.getServerContext method.

Members

All members of **ianywhere.ml.script.ServerContext**, including all inherited members.

addInfoListener method

Syntax

public void **addInfoListener(** LogListener *l* **)**

Remarks

Adds the specified LogListener from the list of listeners to receive a notification when info is printed. The method LogListener.messageLogged (ianywhere.ml.script.ServerContext) is called.

Parameters

- **ll** The LogListener to be notify.

Example

The following code registers a listener of type MyLogListener to receive notifications of info messages.

```
// ServerContext serv_context;
serv_context.addInfoListener(new MyLogListener(ll_out_file));

// The following code shows an example of processing those messages:
class MyLogListener implements LogListener {
    FileOutputStream _out_file;
    public TestLogListener(FileOutputStream out_file) {
        _out_file = out_file;
    }

    public void messageLogged(ServerContext sc, LogMessage msg) {
        String type;
        String user;

        try {
            if (msg.getType() == LogMessage.ERROR) {
                type = "ERROR";
            } else if (msg.getType() == LogMessage.WARNING) {
                type = "WARNING";
            } else if (msg.getType() == LogMessage.INFO) {
                type = "INFO";
            } else {
                type = "UNKNOWN!!!";
            }

            user = msg.getUser();
            if (user == null) {
                user = "NULL";
            }
            _out_file.write(("Caught msg type="
                + type
                + " user=" + user
                + " text=" +msg.getText()
                + "\n").getBytes()
            );
            _out_file.flush();
        }
        catch(Exception e) {

            // if we print the exception from processing an info message,
            // we may recurse indefinitely
            if (msg.getType() != LogMessage.ERROR) {
                // Print some error output to the MobiLink log.
                e.printStackTrace();
            }
        }
    }
}
```

addErrorListener method

Syntax

public void **addErrorListener(** LogListener *ll* **)**

Remarks

Adds the specified LogListener to receive a notification when an error is printed.

When an error is printed, the following method is called LogListener.messageLogged(ianywhere.ml.script.ServerContext, ianywhere.ml.script.LogMessage).

Parameters

- **ll** The LogListener to notify.

See also

- "messageLogged method" on page 555

Example

See: "LogMessage class" on page 555.

addShutdownListener method

Syntax

public void **addShutdownListener(** ShutdownListener *sl* **)**

Remarks

Adds the specified ShutdownListener that is to receive notification before the server context is destroyed. On shutdown, the method ShutdownListener.shutdownPerformed (ianywhere.ml.script.ServerContext) is called.

Parameters

- **sl** The ShutdownListener to notify on shutdown.

Example

See: "ShutdownListener interface" on page 569.

addWarningListener method

Syntax

public void **addWarningListener(** LogListener *ll* **)**

Remarks

Adds the specified LogListener to receive a notification when a warning is printed.

The following method is called: LogListener.messageLogged(ianywhere.ml.script.ServerContext, ianywhere.ml.script.LogMessage).

Parameters

- **ll** The LogListener to notify.

Example

See: "LogMessage class" on page 555.

getProperties method

Syntax

public java.util.Properties **getProperties(**
  java.lang.String *component*,
  java.lang.String *set* **)**

Remarks

Returns the set of properties associated with a given component and property set. These are stored in the MobiLink system table ml_property.

Parameters

- **component** The component.
- **set** The property set.

Returns

The set of properties, which may be empty.

See also

- "ml_add_property system procedure" on page 705

Example

The following code lists all the ServerContext's Properties.

```
import java.util.*;
// ServerContext serverContext;
// PrintStream out
Properties prop = serverContext.getProperties();
prop.list(out);
```

getPropertiesByVersion method

Syntax

public java.util.Properties **getPropertiesByVersion(** java.lang.String *script_version* **)**

Remarks

Returns the set of properties associated with the script version. These are stored in the MobiLink system table ml_property. The script version is stored in the property_set_name column when the component_name is ScriptVersion.

Parameters

- **script_version** The script version for which to return associated properties.

Returns

The set of properties associated with the given script version.

See also

- "ml_add_property system procedure" on page 705

Example

The following code lists all the ServerContext's Properties associated with a given script version.

```
import java.util.*;
// ServerContext serverContext;
// PrintStream out
Properties prop = serverContext.getPropertiesByVersion("MyScriptVersion");
prop.list(out);
```

getPropertySetNames method

Syntax

public Iterator **getPropertySetNames(**
java.lang.String *component_name* **)**

Remarks

Returns the list of property set names for a given component. These are stored in the MobiLink system table ml_property.

Parameters

- **component_name** The name of the component for which to list property names.

Returns

The list of property set names for the given component.

See also

- "ml_add_property system procedure" on page 705

Example

The following code lists all the ServerContext's Properties associated with a given component.

```
import java.util.*;
// ServerContext serverContext;
// PrintStream out
Properties prop = serverContext.getPropertySetNames("Component Name");
prop.list(out);
```

getStartClassInstances method

Gets an array of the start classes that were constructed at server start time. The array length is zero if there are no start classes.

Syntax

public java.lang.Object[ ] **getStartClassInstances( )**

Returns

An array of start classes that were constructed at server start time, or an array of length zero if there are no start classes.

Example

The following is an example that uses getStartClassInstances():

```
Object objs[] = sc.getStartClassInstances();
int i;
for (i=0; i < objs.length; i += 1) {
    if (objs[i] instanceof MyClass) {
        // Use class.
    }
}
```

See also

- "User-defined start classes" on page 530

makeConnection method

Opens and returns a new server connection.

Syntax

public java.sql.Connection **makeConnection( )**
throws **java.sql.SQLException**

Remarks

This connection is owned by the user java code. It must be committed and closed by the user. To access the server context, use DBConnectionContext.getServerContext on the DBConnectionContext for the current connection.

Returns

The new server connection.

Exceptions

- **SQLException** Thrown if an error occurred opening the new connection.

removeErrorListener method

Syntax

public void **removeErrorListener(** LogListener *ll* **)**

Remarks

Removes the specified LogListener from the list of listeners that are to receive a notification when an error is printed.

Parameters

- **ll** The LogListener to no longer notify.

Example

The following code removes a LogListener from the list of error listeners.

```
// ServerContext serverContext;
// LogListener myErrorListener
serverContext.removeErrorListener(myErrorListener);
```

removeInfoListener method

Syntax

public void **removeInfoListener(** LogListener *ll* **)**

Remarks

Remove the specified LogListener from the list of listeners to receive a notification when info is printed.

Parameters

- **ll** The listener to no longer notify.

Example

The following code removes a LogListener from the list of info listeners.

```
// ServerContext serverContext;
// LogListener myInfoListener
serverContext.removeInfoListener(myInfoListener);
```


removeShutdownListener method

Syntax

public void **removeShutdownListener(** ShutdownListener *sl* **)**

Remarks

Removes the specified ShutdownListener from the list of listeners that are to receive notification before the server context is destroyed.

Parameters

- **sl** The ShutdownListener to no longer notify on shutdown.

Example

The following code removes a ShutdownListener from the list of listeners that are to receive notification before the server context is destroyed.

```
// ServerContext serverContext;
// ShutdownListener myShutdownListener
serverContext.removeShutdownListener(myShutdownListener);
```

removeWarningListener method

Syntax

public void **removeWarningListener(** LogListener *ll* **)**

Remarks

Removes the specified LogListener from the list of listeners that are to receive a notification when a warning is printed.

Parameters

- **ll** The LogListener to no longer notify.

Example

The following code removes a LogListener from the list of warning listeners.

```
// ServerContext serverContext;
// LogListener myWarningListener
serverContext.removeWarningListener(myWarningListener);
```

shutdown method

Syntax

public void **shutdown( )**

Remarks

Forces the server to shut down. Any registered ShutdownListener instances have their shutdownPerformed method called.

Example

The following code forces the server to shut down.

```
// ServerContext serverContext;
serverContext.shutdown();
```

ServerException class

Syntax

public **ianywhere.ml.script.ServerException**

Remarks

Thrown to indicate that there is an error condition that makes any further synchronization on the server impossible. Throwing this exception causes the MobiLink server to shut down.

Members

All members of **ianywhere.ml.script.ServerException**, including all inherited members.

- "ServerException constructors" on page 569

Example

The following code is a function that can throw a ServerException if a fatal problem occurs, which causes the MobiLink server to shut down.

```
public void handleUpload(UploadData ud)
    throws SQLException, IOException, ServerException
{

    UploadedTableData tables[] = ud.getUploadedTables();
    if (tables == null) {
        throw new ServerException("Failed to read uploaded tables");
    }

    for (int i = 0; i < tables.length; i++) {
        UploadedTableData currentTable = tables[i];
        println("table " + java.lang.Integer.toString(i)
            + " name: " + currentTable.getName());

        // Print out delete result set.
        println("Deletes");
        printRSInfo(currentTable.getDeletes());

        // Print out insert result set.
        println("Inserts");
        printRSInfo(currentTable.getInserts());

        // print out update result set
        println("Updates");
```



```
            printUpdateRSInfo(currentTable.getUpdates());
        }
    }
```

ServerException constructors

Syntax

public **ServerException( )**

Remarks

Constructs a ServerException with no detail message.

Syntax

public **ServerException(** java.lang.String *s* **)**

Remarks

Constructs a ServerException with a specified detail message.

Parameters

- **s** The detailed message.

Example

See: "ServerException class" on page 568.

ShutdownListener interface

Syntax

public **ianywhere.ml.script.ShutdownListener**

Remarks

The listener interface for catching server shutdowns. Use this interface to ensure that all threads, connections, and other resources are cleaned up before the server exits

Members

All members of **ianywhere.ml.script.ShutdownListener**, including all inherited members.

- "shutdownPerformed method" on page 570

Example

The following code installs a ShutdownListener for the ServerContext.

```
class MyShutdownListener implements ShutdownListener {
    FileOutputStream _outFile;
    public MySutdownListener(FileOutputStream outFile) {
```

```
            _outFile = outFile;
        }

        public void shutdownPerformed(ServerContext sc) {
            // Add shutdown code
            try {
                _outFile.write(("Shutting Down" + "\n").getBytes());
                _outFile.flush();
            }
            catch(Exception e) {
                // Print some error output to the MobiLink log.
                e.printStackTrace();
            }
            // ...
        }
    }
```

The following code registers MyShutdownListener. Call this code from anywhere that has access to the ServerContext such as a class constructor or synchronization script.

```
// ServerContext serv_context;
// FileOutputStream outFile
serv_context.addShutdownListener(new MyShutdownListener(outFile));
```

shutdownPerformed method

Syntax

public void **shutdownPerformed(** ServerContext *sc***)**

Remarks

Invoked before the ServerContext is destroyed due to server shutdown.

Parameters

- **sc** The context for the server that is being shut down.

Example

See: "ShutdownListener interface" on page 569.

SpatialUtilities class

A collection of static methods to work with spatial values.

Syntax

```
public class SpatialUtilities
```

Members

All members of SpatialUtilities class, including all inherited members.

Name	Description
"createSpatialValue method"	Returns a new byte array that contains a spatial value formatted for download: the first four bytes contain the given SRID in little endian, and the remainder is the spatial data passed in the given byte array.
"getBytes method"	Returns a new byte array with the same spatial data as the given byte array, but with the SRID removed.
"getSRID method"	Returns the srid for the given spatial value.
"setSRID method"	Stores the given SRID in the first four bytes of the given byte array.

createSpatialValue method

Returns a new byte array that contains a spatial value formatted for download: the first four bytes contain the given SRID in little endian, and the remainder is the spatial data passed in the given byte array.

Syntax

```
byte[] SpatialUtilities.createSpatialValue(
    int srid,
    byte[] spatial_value
)
```

Parameters

- **srid** The srid
- **spatial_value** The spatial data

Returns

The spatial value formatted for download

getBytes method

Returns a new byte array with the same spatial data as the given byte array, but with the SRID removed.

Syntax

```
byte[] SpatialUtilities.getBytes(byte[] spatial_value)
```

Parameters

- **spatial_value** A spatial value that needs its SRID removed

Returns

The new byte array

getSRID method

Returns the srid for the given spatial value.

Syntax

```
int SpatialUtilities.getSRID(byte[] spatial_value)
```

Parameters

- **spatial_value** The uploaded value. The first four bytes must contain the SRID encoded in little endian.

Returns

The SRID.

setSRID method

Stores the given SRID in the first four bytes of the given byte array.

Syntax

```
void SpatialUtilities.setSRID(byte[] spatial_value, int srid)
```

Parameters

- **spatial_value** The array to store the SRID in
- **srid** The srid to store

SynchronizationException class

Syntax

public **ianywhere.ml.script.SynchronizationException**

Remarks

Thrown to indicate that there is an error condition that makes the completion of the current synchronization impossible. Throwing this exception forces the MobiLink server to rollback.

Members

All members of **ianywhere.ml.script.SynchronizationException**, including all inherited members.

- "SynchronizationException constructors" on page 573

Example

The following code is a function that can throw a SynchronizationException if a problem occurs, which causes the MobiLink server to rollback.


```
public void handleUpload(UploadData ud)
    throws SQLException, IOException, SynchronizationException
{

    UploadedTableData tables[] = ud.getUploadedTables();

    for (int i = 0; i < tables.length; i++) {
        UploadedTableData currentTable = tables[i];
        println("table " + java.lang.Integer.toString(i)
            + " name: " + currentTable.getName());

        // Print out delete result set.
        println("Deletes");
        printRSInfo(currentTable.getDeletes());

        // Print out insert result set.
        println("Inserts");
        printRSInfo(currentTable.getInserts());

        // print out update result set
        println("Updates");
        printUpdateRSInfo(currentTable.getUpdates());

        if (/* Reason for Sync failure */) {
            throw new SynchronizationException("Sync Failed");
        }
    }
}
```

SynchronizationException constructors

Syntax

public **SynchronizationException( )**

Remarks

Constructs a SynchronizationException with no detail message.

Syntax

public **SynchronizationException(** java.lang.String *s* **)**

Remarks

Constructs a SynchronizationException with the specified detail message.

Parameters

- **s** A detail message.

TimestampWithTimeZone class

A java.sql.Timestamp with methods to get and set the time zone.

Syntax

```
public class TimestampWithTimeZone
```

Base classes

- Timestamp

Members

All members of TimestampWithTimeZone class, including all inherited members.

Name	Description
"TimestampWithTimeZone constructor"	Constructs a new TimestampWithTimeZone with the specified year, month, date, hour minute, second, nano, time zone hour and time zone minute.
"equals method"	Returns true if o is equal to the Timestamp portion of this and either o is a TimestampWithTimeZone with the same time zone offset as this, or o is not a TimestampWithTimeZone and this has a time zone offset of 00:00.
"getTimeZoneOffsetHours method"	Gets the hours portion of the time zone offset.
"getTimeZoneOffsetMinutes method"	Gets the minutes portion of the time zone offset.
"setTimeZoneOffsetHours method"	Sets the hours portion of the time zone offset.
"setTimeZoneOffsetMinutes method"	Sets the minutes portion of the time zone offset.
"toString method"	Returns the string representing this timestamp with the form yyyy-mm-dd hh:mm:ss.ffffff Shh:mm, where S is the sign of the hours field.
"toTimestampWithTimeZone method"	Converts the given Timestamp to a TimestampWithTimeZone.
"valueOf method"	Converts a *String* object to a *TimestampWithTimeZone* value.

Remarks

Use this when using the MobiLink direct row API to specify the time zone offset for TIMESTAMP WITH TIME ZONE columns. Note that PreparedStatements and ResultSets from JDBC drivers other than the MobiLink direct row API will treat this as a normal Timestamp.

TimestampWithTimeZone constructor

Constructs a new TimestampWithTimeZone with the specified year, month, date, hour minute, second, nano, time zone hour and time zone minute.

Syntax

```
TimestampWithTimeZone.TimestampWithTimeZone(
    int year,
    int month,
    int date,
    int hour,
    int minute,
    int second,
    int nano,
    int tz_hour,
    int tz_minute
)
```

Parameters

- **year** The year minus 1900
- **month** 0 to 11
- **date** 1 to 31
- **hour** 0 to 23
- **minute** 0 to 59
- **second** 0 to 59
- **nano** 0 to 999,999,999
- **tz_hour** -14 to 14
- **tz_minute** 0 to 59

Exceptions

- **java.lang.IllegalArgumentException** if tz_minute is not in 0 - 59 or tz_hour is not in -14 - 14

equals method

Returns true if o is equal to the Timestamp portion of this and either o is a TimestampWithTimeZone with the same time zone offset as this, or o is not a TimestampWithTimeZone and this has a time zone offset of 00:00.

Overload list

Name	Description
"equals(Object) method"	Returns true if o is equal to the Timestamp portion of this and either o is a TimestampWithTimeZone with the same time zone offset as this, or o is not a TimestampWithTimeZone and this has a time zone offset of 00:00.

Name	Description
"equals(Timestamp) method"	Returns true if o is equal to the Timestamp portion of this and either o is a TimestampWithTimeZone with the same time zone offset as this, or o is not a TimestampWithTimeZone and this has a time zone offset of 00:00.

equals(Object) method

Returns true if o is equal to the Timestamp portion of this and either o is a TimestampWithTimeZone with the same time zone offset as this, or o is not a TimestampWithTimeZone and this has a time zone offset of 00:00.

Syntax

```
boolean TimestampWithTimeZone.equals(Object o)
```

equals(Timestamp) method

Returns true if o is equal to the Timestamp portion of this and either o is a TimestampWithTimeZone with the same time zone offset as this, or o is not a TimestampWithTimeZone and this has a time zone offset of 00:00.

Syntax

```
boolean TimestampWithTimeZone.equals(Timestamp o)
```

Parameters

- **o** The object to compare against

Returns

true if o is equal to this; otherwise false

getTimeZoneOffsetHours method

Gets the hours portion of the time zone offset.

Syntax

```
int TimestampWithTimeZone.getTimeZoneOffsetHours()
```

Returns

the hours portion of the time zone offset

getTimeZoneOffsetMinutes method

Gets the minutes portion of the time zone offset.

Syntax

```
int TimestampWithTimeZone.getTimeZoneOffsetMinutes()
```

Returns

the minutes portion of the time zone offset

setTimeZoneOffsetHours method

Sets the hours portion of the time zone offset.

Syntax

```
void TimestampWithTimeZone.setTimeZoneOffsetHours(int tz_hour)
```

Parameters

- **tz_hour** the new hours portion of the time zone offset

Exceptions

- **java.lang.IllegalArgumentException** if tz_hour is not in -14 - 14

setTimeZoneOffsetMinutes method

Sets the minutes portion of the time zone offset.

Syntax

```
void TimestampWithTimeZone.setTimeZoneOffsetMinutes(int tz_minute)
```

Parameters

- **tz_minute** the new minutes portion of the time zone offset

Exceptions

- **java.lang.IllegalArgumentException** if tz_minute is not in 0 - 59

toString method

Returns the string representing this timestamp with the form yyyy-mm-dd hh:mm:ss.ffffff Shh:mm, where S is the sign of the hours field.

Syntax

```
String TimestampWithTimeZone.toString()
```

Returns

the string representing this timestamp

toTimestampWithTimeZone method

Converts the given Timestamp to a TimestampWithTimeZone.

Syntax

```
TimestampWithTimeZone TimestampWithTimeZone.toTimestampWithTimeZone(
    Timestamp ts
)
```

Parameters

- **ts** The timestamp to convert

Returns

If ts is an instance of TimestampWithTimeZone, returns ts. Otherwise, constructs a new TimestampWithTimeZone that is equivalent to ts with a time zone offset of 00:00.

valueOf method

Converts a *String* object to a *TimestampWithTimeZone* value.

Syntax

```
TimestampWithTimeZone TimestampWithTimeZone.valueOf(String val)
```

Parameters

- **val** timestamp with time zone format in *yyyy-mm-dd hh:mm:ss[.f...][[-+]hh:mm]*. The fractional part may be omitted. The time zone part may be omitted. If the time zone is present, the sign can be omitted.

Returns

The new TimestampWithTimeZone

Exceptions

- **java.lang.IllegalArgumentException** if val does not have the correct format

UpdateResultSet

Syntax

public **ianywhere.ml.script.UpdateResultSet**

Remarks

A result set object including special methods for accessing the pre-image (old) and post-image (new) values of a specified row. To obtain an UpdateResultSet instance, use the DownloadTableData.getUpdates method.

UpdateResultSet extends java.sql.ResultSet and adds the setNewRowValues and setOldRowValues methods. Otherwise it can be used as a regular resultset. Consult your Java Software Development Kit documentation for more information about java.sql.ResultSet

See also

- DownloadTableData "getUpdates method" on page 585
- "Handling direct upload conflicts" on page 676
- "Direct row handling" on page 671

Members

All members of **ianywhere.ml.script.UpdateResultSet**, including all inherited members.

- "setNewRowValues method" on page 579
- "setOldRowValues method" on page 580

Example

The following code shows how to obtain an UpdateResultSet instance from a DownloadTableData instance.

```
// DownloadTableData tableData
UpdateResultSet results = tableData.getUpdates();
```

setNewRowValues method

Syntax

public void **setNewRowValues( )**

Remarks

Sets the mode of this result set to return new column values (the post update row). The result set represents the latest updated values in the remote client database. This is the default mode.

See also

- "UpdateResultSet" on page 578
- "Handling direct upload conflicts" on page 676
- "Direct row handling" on page 671

Example

The following code shows how to set the mode of the UpdateResultSet to return new column values.

```
// UpdateResultSet results
results.setNewRowValues();
```

setOldRowValues method

Syntax

public void **setOldRowValues( )**

Remarks

Sets the mode of this result set to return old column values (the pre update row). In this mode, the UpdateResultSet represents old column values obtained by the client in the last synchronization.

See also

- "UpdateResultSet" on page 578
- "Handling direct upload conflicts" on page 676
- "Direct row handling" on page 671

Example

The following code shows how to set the mode of the UpdateResultSet to return old column values.

```
// UpdateResultSet results
results.setOldRowValues();
```

UploadData interface

Syntax

public **ianywhere.ml.script.UploadData**

Remarks

Encapsulates upload operations for direct row handling. An UploadData instance representing a single upload transaction is passed to the handle_UploadData synchronization event.

Caution
You must handle direct row handling upload operations in the method registered for the handle_UploadData event. The UploadData is destroyed after each call to the registered method. Do not create a new instance of UploadData to use in subsequent events.

Use the UploadData.getUploadedTables or UploadData.getUploadedTableByName methods to obtain UploadedTableData instances.

A synchronization has one UploadData unless the remote database is using transactional upload.

See also

- "UploadedTableData interface" on page 582
- "handle_UploadData connection event" on page 394
- "Direct row handling" on page 671
- "Handling direct uploads" on page 675

Members

All members of **ianywhere.ml.script.UploadData**, including all inherited members.

- "getUploadedTableByName method" on page 581
- "getUploadedTables method" on page 582

Example

See "handle_UploadData connection event" on page 394.

getUploadedTableByName method

Syntax

public UploadedTableData **getUploadedTableByName(**
  java.lang.String *table_name*
)

Remarks

Returns a UploadedTableData instance representing the specified table.

Parameters

- **table_name** The name of the uploaded table for which you want the uploaded data.

Returns

An UploadedTableData instance representing the specified table, or null if a table of the given name does not exist for the current synchronization.

See also

- "UploadData interface" on page 580
- "UploadedTableData interface" on page 582
- "Direct row handling" on page 671

Example

Assume you use a method called HandleUpload for the handle_UploadData synchronization event. The following example uses the getUploadedTableByName method to return an UploadedTableData instance for the remoteOrders table.

```
// The method used for the handle_UploadData event.

public void handleUpload(UploadData ut)
    throws SQLException, IOException
{

    UploadedTableData uploaded_t1 =
ut.getUploadedTableByName("remoteOrders");
    // ...
}
```

getUploadedTables method

Syntax

public UploadedTableData[] **getUploadedTables( )**

Remarks

Returns an array of UploadedTableData objects for the current synchronization. The order to the tables in the array is the same order that MobiLink uses for SQL row handling, and is the optimal order for preventing referential integrity violations. If your data source is a relational database, use this table order.

Returns

An array of UploadedTableData objects for the current synchronization. The order of tables in the array is the same as the upload order of the client.

See also

- "UploadData interface" on page 580
- "UploadedTableData interface" on page 582
- "Direct row handling" on page 671

Example

Assume you use a method called HandleUpload for the handle_UploadData synchronization event. The following example uses the getUploadedTables method to return UploadedTableData instances for the current upload transaction.

```
// The method used for the handle_UploadData event.

public void handleUpload(UploadData ud)
    throws SQLException, IOException
{
    UploadedTableData tables[] = ud.getUploadedTables();
    //...
}
```

UploadedTableData interface

Syntax

public **ianywhere.ml.script.UploadedTableData**

Remarks

Encapsulates table operations for direct row handling uploads. You can use an UploadedTableData instance to obtain a table's insert, update, and delete operations for a single upload transaction. Use the UploadedTableData.getInserts, UploadedTableData.getUpdates, and UploadedTableData.getDeletes methods to return standard JDBC java.sql.ResultSet objects.

Consult your Java Software Development Kit documentation for more information about java.sql.ResultSet and java.sql.ResultSetMetaData.

Table metadata can be accessed using the UploadedTableData.getMetaData method or the result sets returned by getInserts, getUpdates, and getDeletes. The delete result set only includes primary key columns for a table.

See also

- "UploadData interface" on page 580
- "handle_UploadData connection event" on page 394
- "Direct row handling" on page 671

Members

All members of **ianywhere.ml.script.UploadedTableData**, including all inherited members.

- "getDeletes method" on page 583
- "getInserts method" on page 584
- "getUpdates method" on page 585
- "getName method" on page 586
- "getMetaData method" on page 586

Example

See: "UploadData interface" on page 580.

getDeletes method

Syntax

public java.sql.ResultSet **getDeletes( )**

Remarks

Returns a java.sql.ResultSet object representing delete operations uploaded by a MobiLink client. The result set contains primary key values for deleted rows.

Returns

A java.sql.ResultSet object that represents delete operations uploaded by a MobiLink client.

See also

- "UploadedTableData interface" on page 582
- "handle_UploadData connection event" on page 394
- "Direct row handling" on page 671

Example

Assume your remote client contains a table called remoteOrders. The following example uses the DownloadTableData.getDeletes method to obtain a result set of deleted rows. In this case, the delete result set includes a single primary key column.

```
import ianywhere.ml.script.*;
import java.sql.*;
```

```
// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut)
    throws SQLException, IOException
{
    // Get an UploadedTableData for the remoteOrders table.
    UploadedTableData remoteOrdersTable =
ut.getUploadedTableByName("remoteOrders");

    // Get deletes uploaded by the MobiLink client.
    java.sql.ResultSet delete_rs = remoteOrdersTable.getDeletes();

     while (delete_rs.next()) {
         // Get primary key values for deleted rows.
         int deleted_id = delete_rs.getInt(1);

         // ...
    }
    delete_rs.close();
}
```

getInserts method

Syntax

public java.sql.ResultSet **getInserts( )**

Remarks

Returns a java.sql.ResultSet object representing insert operations uploaded by a MobiLink client. Each insert is represented by one row in the result set.

Returns

A java.sql.ResultSet object representing insert operations uploaded by a MobiLink client.

See also

- "UploadedTableData interface" on page 582
- "Direct row handling" on page 671

Example

Assume your remote client contains a table called remoteOrders. The following example uses the DownloadTableData.getInserts method to obtain a result set of inserted rows. The code obtains the order amount for each row in the current upload transaction.

```
import ianywhere.ml.script.*;
import java.sql.*;

// The method used for the handle_UploadData event
public void HandleUpload(UploadData ut)
    throws SQLException, IOException
{
    // Get an UploadedTableData instance representing the remoteOrders table.
    UploadedTableData remoteOrdersTable =
ut.getUploadedTableByName("remoteOrders");

    // Get inserts uploaded by the MobiLink client.
```



```
        java.sql.ResultSet rs = remoteOrdersTable.getInserts();
        while (rs.next()) {
            // get the uploaded order_amount
            double order_amount = rs.getDouble("order_amount");

            // ...
        }
        rs.close();
    }
```

getUpdates method

Syntax

public UpdateResultSet **getUpdates( )**

Remarks

Returns a UpdateResultSet object representing update operations uploaded by a MobiLink client. Each update is represented by one row including all column values. UpdateResultSet extends java.sql.ResultSet to include special methods for MobiLink conflict detection.

Returns

An UpdateResultSet object representing update operations uploaded by a MobiLink client.

See also

- "UploadedTableData interface" on page 582
- "UpdateResultSet" on page 578
- "handle_UploadData connection event" on page 394
- "Handling direct upload conflicts" on page 676
- "Direct row handling" on page 671

Example

Assume your remote client contains a table called remoteOrders. The following example uses the UploadedTableData.getUpdates method to obtain a result set of updated rows. The code obtains the order amount for each row.

```
import ianywhere.ml.script.*;
import java.sql.*;

// the method used for the handle_UploadData event
public void HandleUpload(UploadData ut)
    throws SQLException, IOException
{
    // Get an UploadedTableData instance representing the remoteOrders table.
    UploadedTableData remoteOrdersTable =
ut.getUploadedTableByName("remoteOrders");

    // Get inserts uploaded by the MobiLink client.
    java.sql.ResultSet rs = remoteOrdersTable.getUpdates();
    while (rs.next()) {
        // Get the uploaded order_amount.
        double order_amount = rs.getDouble("order_amount");
```

```
            // ...
        }
        rs.close();
    }
```

getName method

Syntax

public java.lang.String **getName( )**

Remarks

Returns the table name for the UploadedTableData instance. You can also access the table name using the java.sql.ResultSetMetaData instance returned by the getMetaData method.

Returns

The table name for the UploadedTableData instance.

See also

- "UploadedTableData interface" on page 582
- UploadedTableData "getMetaData method" on page 586
- "handle_UploadData connection event" on page 394
- "Direct row handling" on page 671

Example

The following example obtains the name of each uploaded table in a single upload transaction.

```
import ianywhere.ml.script.*;
import java.sql.*;

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ud) {
    throws SQLException, IOException
{
    int i;

    // Get UploadedTableData instances.
    UploadedTableData tables[] = ud.getUploadedTables();

    for (i=0; i<tables.length; i+=1) {
        // Get the table name.
        String table_name = tables[i].getName();

        // ...
    }
}
```

getMetaData method

Syntax

public java.sql.ResultSetMetaData **getMetaData( )**

Remarks

Gets the metadata for the UploadedTableData instance. The metadata is a standard java.sql.ResultSetMetaData instance.

If you want the ResultSetMetaData to contain column name information, you must specify the client option to send column names.

Consult your Java Software Development Kit documentation for more information about java.sql.ResultSetMetaData.

Returns

The metadata for the UploadedTableData instance.

See also

- dbmlsync: "SendColumnNames (scn) extended option" [*MobiLink - Client Administration*]
- UltraLite: "Send Column Names synchronization parameter" [*UltraLite - Database Management and Reference*]

Example

The following example obtains a java.sql.ResultSetMetaData instance for an uploaded table called remoteOrders. The code uses the ResultSetMetaData.getColumnCount and getColumnLabel methods to compile a list of column names.

```
import ianywhere.ml.script.*;
import java.sql.*;

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {
    throws SQLException, IOException
{
    // Get an UploadedTableData instance representing the remoteOrders table.
    UploadedTableData remoteOrdersTable =
ut.getUploadedTableByName("remoteOrders");

    // get inserts uploaded by the MobiLink client
    java.sql.ResultSet rs = remoteOrdersTable.getInserts();

    // Obtain the result set metadata.
    java.sql.ResultSetMetaData md = rs.getMetaData();
    String columnHeading = "";

    // Compile a list of column names.
    for (int c=1; c <= md.getColumnCount(); c += 1) {
        columnHeading += md.getColumnLabel( c );

        if (c < md.getColumnCount()) {
            columnHeading += ", ";
        }
    }
    //...
}
```

In this case, a method called HandleUpload handles the handle_UploadData synchronization event.

See also

- "UploadedTableData interface" on page 582
- "Direct row handling" on page 671
- "SendColumnNames (scn) extended option" [*MobiLink - Client Administration*]
- "handle_UploadData connection event" on page 394

Writing synchronization scripts in .NET

MobiLink supports the Visual Studio programming languages for writing synchronization scripts. To write MobiLink scripts in .NET, you can use any language that lets you create valid .NET assemblies. In particular, the following languages are tested and documented:

- C#
- Visual Basic .NET
- C++

.NET synchronization logic can function just as SQL logic functions: the MobiLink server can make calls to .NET methods on the occurrence of MobiLink events just as it accesses SQL scripts on the occurrence of MobiLink events. A .NET method can return a SQL string to MobiLink.

This section tells you how to set up, develop, and run .NET synchronization logic for C#, Visual Basic .NET, and C++. It includes a sample application and the MobiLink server API for .NET Reference.

See also

- "Tutorial: Using .NET synchronization logic" [*MobiLink - Getting Started*]
- "Options for writing server-side synchronization logic" [*MobiLink - Getting Started*]
- "Writing synchronization scripts" on page 263

Setting up .NET synchronization logic

When you implement synchronization scripts in .NET, you must tell MobiLink where to find the packages, classes, and methods that are contained in your assemblies.

To implement synchronization scripts in .NET

1. Create your own class or classes. Write a method for each required synchronization event. These methods must be public.

   For more information about methods, see "Methods" on page 593.

   Each class with non-static methods should have a public constructor. The MobiLink server automatically instantiates each class the first time a method in that class is called for a connection.

See "Constructors" on page 592.

2. Create one or more assemblies. While compiling, reference *iAnywhere.MobiLink.Script.dll*, which contains a repository of MobiLink server API classes to use in your own .NET methods. *iAnywhere.MobiLink.Script.dll* is located in *install-dir\Assembly\V2*.

   You can compile your class on the command line, or using Visual Studio or another .NET development environment.

   See "MobiLink server .NET API reference (.NET 2.0)" on page 602.

3. Compile your project.

   For example, compile from Visual Studio as follows:

   a. On the **VS.NET Project** menu, choose **Add Existing Item**.

   b. Locate *iAnywhere.MobiLink.Script.dll*.

      In the **Open** list, choose **Link File**.

      Note
      For Visual Studio, always use the Link File method. Do not use the Add Reference option to reference *iAnywhere.MobiLink.Script.dll*. The Add Reference option duplicates *iAnywhere.MobiLink.Script.dll* in the same physical directory as your class assembly, creating problems for the MobiLink server.

   c. Use the **Build** menu to build your assembly.

   You can also compile from the command line, as follows:

   Replace *dll-path* with the path to *iAnywhere.MobiLink.Script.dll*. for example, in C#:

   ```
   csc /out:dll-pathout.dll /target:library /reference:dll-
   pathiAnywhere.MobiLink.Script.dll sync_v1.cs
   ```

4. In the MobiLink system tables in your consolidated database, specify the name of the package, class, and method to call for each synchronization script. No more than one class is permitted per script version.

   For example, you can add this information to the MobiLink system tables using the ml_add_dnet_connection_script stored procedure or the ml_add_dnet_table_script stored procedure. The following SQL statement, when run in a SQL Anywhere database, specifies that myNamespace.myClass.myMethod should be run whenever the authenticate_user connection-level event occurs.

   ```
   CALL ml_add_dnet_connection_script(
       'version1',
       'authenticate_user',
       'myNamespace.myClass.myMethod'
   )
   ```

   Note
   The fully qualified method name is case sensitive.

As a result of this procedure call, the script_language column of the ml_script system table contains the word **dnet**. The script column contains the qualified name of a public .NET method.

See "ml_add_dnet_connection_script system procedure" on page 695 and "ml_add_dnet_table_script system procedure" on page 697.

You can also add this information using Sybase Central.

See "Adding and deleting scripts" on page 285.

5. Instruct the MobiLink server to load assemblies and start the CLR. You tell MobiLink where to locate these assemblies using options in the mlsrv12 command line. There are two options to choose from:

   - **Use -sl dnet (-MLAutoLoadPath)** This sets the given path to the application base directory and loads all the private assemblies within it. This is usually the preferred option. For example, to load all assemblies located in *dll-path*, enter:

     ```
     mlsrv12 -c "dsn=consolidated1" -sl dnet(-MLAutoLoadPath=dll-path)
     ```

     When you use the -MLAutoLoadPath option you cannot specify a domain when entering the fully qualified method name for the event script.

     See "Loading assemblies" on page 598 and "-sl dnet mlsrv12 option" on page 64.

   - **Use -sl dnet (-MLDomConfigFile)** This option requires a configuration file that contains domain and assembly settings. You should use this option when you have shared assemblies, when you don't want to load all the assemblies in a directory, or when for some other reason you need to use a configuration file.

     For more information about loading shared assemblies, see "Loading assemblies" on page 598. For more information about the mlsrv12 option -sl dnet, see "-sl dnet mlsrv12 option" on page 64.

   Note
   You can use the -MLAutoLoadPath option or the -MLDomConfigFile option, but not both.

Writing .NET synchronization logic

To write .NET synchronization logic, you require knowledge of MobiLink events, some knowledge of .NET, and familiarity with the MobiLink server API for .NET.

For a complete description of the API, see "MobiLink server .NET API reference (.NET 2.0)" on page 602.

.NET synchronization logic can be used to maintain state information, and implement logic around the upload and download events. For example, a begin_synchronization script written in .NET could store the MobiLink user name in a variable. Scripts called later in the synchronization process can access this variable. Also, you can use .NET to access rows in the consolidated database, before or after they are committed.

Using .NET also reduces dependence on the consolidated database. Behavior is affected less by upgrading the consolidated database to a new version or switching to a different database management system.

Direct row handling

You can use MobiLink direct row handling to communicate remote data to any central data source, application, or web service. Direct row handling uses special classes in the MobiLink server APIs for Java or .NET for direct access to synchronized data.

See "Direct row handling" on page 671.

Class instances

The MobiLink server instantiates your classes at the database connection level. When an event is reached for which you have written a non-static .NET method, the MobiLink server automatically instantiates the class, if it has not already done so on the present database connection.

See "Constructors" on page 592.

Note
All methods directly associated with a connection-level or table-level event for one script version must belong to the same class.

For each database connection, once a class has been instantiated, the class persists until that connection is closed. So, the same instance may be used for multiple consecutive synchronization sessions. Unless it is explicitly cleared, information present in public or private variables persists across synchronizations that occur on the same connection.

You can also use static classes or variables. In this case, the values are available across all connections in the same domain.

The MobiLink server automatically deletes your class instances only when the connection to the consolidated database is closed.

Transactions

The normal rules regarding transactions apply to .NET methods. The start and duration of database transactions is critical to the synchronization process. Transactions must be started and ended only by the MobiLink server. Explicitly committing or rolling back transactions on the synchronization connection within a .NET method violates the integrity of the synchronization process and can cause errors.

These rules apply only to the database connections created by the MobiLink server and, in particular, to SQL statements returned by methods.

SQL-.NET data types

The following table shows SQL data types and the corresponding .NET data types for MobiLink script parameters.

SQL data type	Corresponding .NET data type
VARCHAR	string
CHAR	string
INTEGER	int
BIGINT	long
BINARY	byte [ ]
TIMESTAMP	DateTime
INOUT INTEGER	ref int
INOUT VARCHAR	ref string
INOUT CHAR	ref string
INOUT BYTEARRAY	ref byte [ ]
INOUT TIMESTAMP	ref DateTime

Constructors

The constructor of your class takes no parameters or takes one iAnywhere.MobiLink.Script.DBConnectionContext parameter. For example:

```
public ExampleClass(iAnywhere.MobiLink.Script.DBConnectionContext cc)
```

or

```
public ExampleClass()
```

The synchronization context passed to you is for the connection through which the MobiLink server is synchronizing the current user.

The DBConnectionContext.GetConnection method returns the same database connection that MobiLink is using to synchronize the present user. You can execute statements on this connection, but you must not commit or roll back the transaction. The MobiLink server manages the transactions.

The MobiLink server uses the constructor that takes a iAnywhere.MobiLink.Script.DBConnectionContext parameter if it exists. If it does not, it uses the void constructor.

See “DBConnectionContext interface” on page 614.

Methods

In general, you implement one method for each synchronization event. These methods must be public. If they are private, the MobiLink server cannot use them and fails to recognize that they exist.

The names of the methods are not important, as long as the names match the names specified in the ml_script table in the consolidated database. In the examples included in the documentation, however, the method names are the same as those of the MobiLink events. This naming convention makes the .NET code easier to read.

The signature of your method should match the signature of the script for that event, except that you can truncate the parameter list if you do not need the values of parameters at the end of the list. You should accept only the parameters you need, because overhead is associated with passing the parameters.

You cannot, however, overload the methods. Only one method prototype per class may appear in the ml_script system table.

Registering values

Methods called for a SQL-based upload or download must return a valid SQL language statement. The return type of these methods must be String. No other return types are allowed.

The return type of all other scripts must either be string or void. No other types are allowed. If the return type is a string and not null, the MobiLink server assumes that the string contains a valid SQL statement and executes this statement in the consolidated database as it would an ordinary SQL-language synchronization script. If a method ordinarily returns a string but does not want to execute a SQL statement against the database upon its return, it can return null.

User-defined start classes

You can define start classes that are loaded automatically when the server is started. The purpose of this feature is to allow you to write .NET code that executes at the time the MobiLink server starts the CLR—before the first synchronization. This means you can create connections or cache data before the first user synchronization request in the server instance.

You do this with the MLStartClasses option of the mlsrv12 -sl dnet option. For example, the following is part of an mlsrv12 command line. It causes mycl1 and mycl2 to be loaded as start classes.

```
-sl dnet(-MLStartClasses=MyNameSpace.MyClass.mycl1,MyNameSpace.MyClass.mycl2)
```

Classes are loaded in the order in which they are listed. If the same class is listed more than once, more than one instance is created.

All start classes must be public and must have a public constructor that either accepts no arguments or accepts one argument of type MobiLink.Script.ServerContext.

The names of loaded start classes are output to the MobiLink log with the message "Loaded .NET start class: *classname*".

For more information about .NET CLR, see "-sl dnet mlsrv12 option" on page 64.

To see the start classes that are constructed at server start time, see "GetStartClassInstances method" on page 650.

Example

The following is a start class template. It starts a daemon thread that processes events and creates a database connection. (Not all start classes need to create a thread but if a thread is spawned it should be a daemon thread.)

```
using System;
using System.IO;
using System.Threading;
using iAnywhere.MobiLink.Script;

namespace TestScripts {
    public class MyStartClass {
        ServerContext    _sc;
        bool             _exit_loop;
        Thread           _thread;
        OdbcConnection   _conn;

        public MyStartClass(ServerContext sc) {

            // Perform setup first so that an exception
            // causes MobiLink startup to fail.
            _sc = sc;

            // Create connection for use later.
            _conn = _sc.makeConnection();
            _exit_loop = false;
            _thread = new Thread(new ThreadStart(run)) ;
            _thread.IsBackground = true;
            _thread.Start();
        }

        public void run() {
            ShutdownCallback callback = new
ShutdownCallback(shutdownPerformed);
            _sc.ShutdownListener += callback;

            // run() can't throw exceptions.
            try {
                handlerLoop();
                _conn.close();
                _conn = null;
            }
            catch(Exception e) {
                // Print some error output to the MobiLink log.
                Console.Error.Write(e.ToString());

                // There is no need to be notified of shutdown.
                _sc.ShutdownListener -= callback;

                // Ask server to shut down so this fatal error can be fixed.
                _sc.Shutdown();
            }

            // Shortly after return, this thread no longer exists.
            return;
        }
```



```
        public void shutdownPerformed(ServerContext sc) {
            // Stop the event handler loop.
            try {
                _exit_loop = true;

                // Wait a maximum of 10 seconds for thread to die.
                _thread.Join(10*1000);
            }
            catch(Exception e) {
                // Print some error output to the MobiLink log.
                Console.Error.Write(e.ToString());
            }
        }

        private void handlerLoop() {
            while (!_exit_loop) {
                // Handle events in this loop.
                Thread.Sleep(1*1000);
            }
        }
    }
}
```

Printing information from .NET

You may choose to add statements to your .NET methods that print information to the MobiLink log using System.Console. Doing so can help you track the progress and behavior of your classes.

Performance tip
Printing information in this manner to the MobiLink log is a useful monitoring tool, but is not recommended in a production scenario.

The same technique can be exploited to log arbitrary synchronization information or collect statistical information on how your scripts are used.

Handling MobiLink server errors with .NET

When scanning the log is not enough, you can monitor your applications programmatically. For example, you can send messages of a certain type in an email.

You can write methods that are passed a class representing every error or warning message that is printed to the log. This may help you monitor and audit a MobiLink server.

The following code installs a listener for all error messages and prints the information to a StreamWriter.

```
class TestLogListener {
    public TestLogListener(StreamWriter output_file) {
        _output_file    = output_file;
    }

    public void errCallback(ServerContext sc, LogMessage lm) {
        string type;
```

```
            string user;

            if (lm.Type == LogMessage.MessageType.ERROR) {
                type = "ERROR";
            } else if (lm.Type==LogMessage.MessageType.WARNING) {
                type = "WARNING";
            }
            else {
                type = "INVALID TYPE!!";
            }
            if (lm.User == null) {
                user = "null";
            }
            else {
                user = lm.User;
            }

            _output_file.WriteLine("Caught msg type=" + type
                + " user=" + user
                + " text=" + lm.Text);
            _output_file.Flush();
        }
        StreamWriter _output_file;
}
```

The following code registers the TestLogListener. Call this code from somewhere that has access to the ServerContext such as a class constructor or synchronization script.

```
// ServerContext serv_context;
TestLogListener errtll = new TestLogListener(log_listener_file);
serv_context.ErrorListener += new LogCallback(errtll.errCallback);
```

See also

- "LogCallback delegate" on page 667
- "ErrorListener event" on page 651
- "WarningListener event" on page 652
- "LogMessage class" on page 643
- "MessageType enumeration" on page 644

Debugging .NET synchronization logic

The following procedure describes one way you can debug your .NET scripts using Visual Studio.

To debug .NET scripts

1. Compile your code with debugging information turned on using one of the following methods:
   - On the csc command line, set the /debug+ option.
   - Use Microsoft Visual Studio settings to set debug output.
     - Choose **File** » **Build** » **Configuration Manager**.

       In the **Active Solution Configuration** list, choose **Debug**.
     - Build your assembly.

2. Close running instances of Visual Studio that contain your source files.

3. In this step, you start a new Visual Studio instance to debug the MobiLink server and your .NET synchronization scripts. Start Visual Studio using a command line option to debug the MobiLink server.
   - At a command prompt, navigate to the *Common7\IDE* subdirectory of your Visual Studio installation.
   - Start devenv (the Visual Studio IDE) using the /debugexe option.

     For example, run the following command to debug the MobiLink server. Remember to specify mlsrv12 options, including the connection string and the option to load .NET assemblies.

     For 32-bit Windows environments:

     ```
     devenv /debugexe %sqlany12%\bin32\mlsrv12.exe -c ...
     ```

     For 64-bit Windows environments:

     ```
     devenv /debugexe %sqlany12%\bin64\mlsrv12.exe -c ...
     ```

     Visual Studio starts and *mlsrv12.exe* appears in the Solution Explorer window.

4. Set up Microsoft Visual Studio for debugging .NET code:
   - In the Visual Studio Solution Explorer window, right-click *mlsrv12.exe* and choose Properties.
   - Change Debugger Type from Auto to Mixed or Managed Only to ensure that Visual Studio only debugs your .NET synchronization scripts.

5. Open the associated .NET source files and set break points.

   Open the source files individually in the mlsrv12 solution. Do not open the original solution or project file.

6. Start MobiLink from the Debug menu or by pressing F5.

   If prompted, save *mlsrv12.sln*.

   If the No Symbolic Information window appears, click **OK** to debug anyway. You are debugging the managed .NET synchronization scripts that MobiLink calls, not the MobiLink server itself.

7. Perform a synchronization that causes the code with a breakpoint to be executed by MobiLink.

.NET synchronization techniques

This section describes techniques you can use to tackle common .NET synchronization tasks.

Uploading or downloading rows

For information about how to upload or download rows via .NET, see "Direct row handling" on page 671.

Loading shared assemblies

This section details options to load .NET assemblies and details the process to load shared assemblies.

Loading assemblies

A .NET assembly is a package of types, metadata, and executable code. In .NET applications, all code must be in an assembly. Assembly files have the extension *.dll* or *.exe*.

There are the following types of assemblies:

- **Private assemblies** A private assembly is a file in the file system.
- **Shared assemblies** A shared assembly is an assembly that is installed in the global assembly cache.

Before MobiLink can load a class and call a method of that class, it must locate the assembly that contains the class. MobiLink only needs to locate the assembly that it calls directly. The assembly can then call any other assemblies it needs.

For example, MobiLink calls MyAssembly, and MyAssembly calls UtilityAssembly and NetworkingUtilsAssembly. In this situation, MobiLink only needs to be configured to find MyAssembly.

MobiLink provides the following ways to load assemblies:

- **Use -sl dnet (-MLAutoLoadPath)** This option only works with private assemblies. It sets the path to the application base directory and loads all the assemblies within it.

  When you use the -MLAutoLoadPath option you cannot specify a domain when entering the fully qualified method name for the event script.

  When you specify a path and directory with -MLAutoLoadPath, MobiLink does the following:

  - sets this path as the application base path
  - loads all classes in all files ending with *.dll* or *.exe* in the directory that you specified
  - creates one application domain and loads into that domain all user classes that do not have a domain specified

  Assemblies in the global assembly cache cannot be called directly with this option. To call these shared assemblies, use -MLDomConfigFile.

- **Use -sl dnet (-MLDomConfigFile)** This option works with both private and shared assemblies. It requires a configuration file that contains domain and assembly settings. You should use this option when you have shared assemblies, when you don't want to load all the assemblies in the application base path, or when for some other reason you need to use a configuration file.

  With this option, MobiLink reads the settings in the specified domain configuration file. A domain configuration file contains configuration settings for one or more .NET domains. If there is more than

one domain represented in the file, the first one that is specified is used as the default domain. (The default domain is used when scripts do not have a domain specified.)

When loading assemblies, MobiLink tries to load the assembly first as private, and then attempts to load the assembly from the global assembly cache. Private assemblies must be located in the application base directory. Shared assemblies are loaded from the global assembly cache.

With the -MLDomConfigFile option, only assemblies that are specified in the domain configuration file can be called directly from event scripts.

Sample domain configuration file

A sample domain configuration file called *mlDomConfig.xml* is installed with MobiLink. You can write your own file from scratch, or edit the sample to suit your needs. The sample file is located in the SQL Anywhere path, in

MobiLink\setup\dnet\mlDomConfig.xml

The following is the content of the sample domain configuration file *mlDomConfig.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="iAnywhere.MobiLink.mlDomConfig"
xsi:schemaLocation='iAnywhere.MobiLink.mlDomConfig mlDomConfig.xsd'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' >
    <domain>
        <name>SampleDomain1</name>
        <appBase>C:\scriptsDir</appBase>
        <configFile></configFile>
        <assembly name="Assembly1" />
        <assembly name="Assembly2" />
    </domain>

    <domain>
        <name>SampleDomain2</name>
        <appBase>\Dom2assembly</appBase>
        <configFile>\Dom2assembly\AssemblyRedirects.config</configFile>
        <assembly name="Assembly3" />
        <assembly name="Assembly4" />
    </domain>
</config>
```

The following is an explanation of the contents of *mlDomConfig.xml*:

- **name** is the domain name, used when specifying the domain in an event script. An event script with the format `"DomainName:Namespace.Class.Method"` would require that the DomainName domain be in the domain configuration file.

  You must specify at least one domain name.

- **appBase** is the directory that the domain should use as its application base directory. All private assemblies are loaded by the .NET CLR based on this directory. You must specify appBase.

- **configFile** is the .NET application configuration file that should be used for the domain. This can be left blank. It is usually used to modify the default assembly binding and loading behavior. Refer to your .NET documentation for more information about application configuration files.

- **assembly** is the name of an assembly that MobiLink should load and search when resolving type references in event scripts. You must specify at least one assembly. If an assembly is used in more than one domain, it must be specified as an assembly in each domain. If the assembly is private, it must be in the application base directory for the domain.

For more information about the mlsrv12 option -sl dnet, see "-sl dnet mlsrv12 option" on page 64.

.NET synchronization example

This example modifies an existing application to describe how to use .NET synchronization logic to handle the authenticate_user event. It creates a C# script for authenticate_user called *AuthUser.cs*. This script looks up the user's password in a table called user_pwd_table and authenticates the user based on that password.

To create your .NET synchronization script

1. Add the table user_pwd_table to the database. Execute the following SQL statements in Interactive SQL:

```
CREATE TABLE user_pwd_table (
  user_name  varchar(128) PRIMARY KEY NOT NULL,
  pwd        varchar(128)
)
```

2. Add a user and password to the table:

```
INSERT INTO user_pwd_table VALUES('user1', 'myPwd')
```

3. Create a directory for your .NET assembly. For example, *c:\mlexample*.

4. Create a file called *AuthUser.cs* with the following contents:

   See "authenticate_user connection event" on page 315.

```
using System;
using iAnywhere.MobiLink.Script;
namespace MLExample {

public class AuthClass {
    private DBConnection  _conn;

    /// AuthClass constructor.
    public AuthClass(DBConnectionContext cc) {
        _conn   = cc.GetConnection();
    }

    /// The DoAuthenticate method handles the 'authenticate_user'
    /// event.
    /// Note: This method does not handle password changes for
    /// advanced authorization status codes.
    public void DoAuthenticate(
        ref int authStatus,
        string user,
        string pwd,
        string newPwd)
    {
```



```
            DBCommand   pwd_command = _conn.CreateCommand();
            pwd_command.CommandText = "select pwd from user_pwd_table"
                + " where user_name = ? ";
            pwd_command.Prepare();

            // Add a parameter for the user name.
            DBParameter user_param  = new DBParameter();
            user_param.DbType       = SQLType.SQL_CHAR;

            // Set the size for SQL_VARCHAR.
            user_param.Size     = (uint) user.Length;
            user_param.Value    = user;
            pwd_command.Parameters.Add(user_param);

            // Fetch the password for this user.
            DBRowReader rr      = pwd_command.ExecuteReader();
            object[] pwd_row    = rr.NextRow();

            if (pwd_row == null) {
                // User is unknown.
                authStatus  = 4000;
            }
            else {
                if (((string) pwd_row[0]) == pwd) {
                    // Password matched.
                    authStatus = 1000;
                }
                else {
                    // Password did not match.
                    authStatus = 4000;
                }
            }
            pwd_command.Close();
            rr.Close();
            return;
        }
    }
```

The MLExample.AuthClass.DoAuthenticate method handles the authenticate_user event. It accepts the user name and password and returns an authorization status code indicating the success or failure of the validation.

5. Compile the file *AuthUser.cs*. You can do this on the command line or in Visual Studio.

   For example, the following command line compiles *AuthUser.cs* and generate an Assembly named *example.dll* in *c:\mlexample*.

   ```
   csc /out:c:\mlexample\example.dll /target:library /reference:"%SQLANY12%
   \Assembly\V2\iAnywhere.MobiLink.Script.dll" AuthUser.cs
   ```

6. Register .NET code for the authenticate_user event. The method you need to execute (DoAuthenticate) is in the namespace MLExample and class AuthClass. Execute the following SQL:

   ```
   CALL ml_add_dnet_connection_script('ex_version', 'authenticate_user',
   'MLExample.AuthClass.DoAuthenticate');
   COMMIT
   ```

7. Run the MobiLink server with the following option. This option causes MobiLink to load all assemblies in *c:\myexample*:

```
-sl dnet (-MLAutoLoadPath=c:\mlexample)
```

Now, when a user synchronizes with the version ex_version, they are authenticated with the password from the table user_pwd_table.

MobiLink server .NET API reference (.NET 2.0)

This section explains the MobiLink .NET interfaces and classes, and their associated methods, properties, and constructors. To use these classes, reference the *iAnywhere.MobiLink.Script.dll* assembly, located in *install-dir\Assembly\V2*.

Namespace

```
iAnywhere.MobiLink.Script
```

DateTimeWithTimeZone class

Represents a DateTime with time zone offsets.

Visual Basic syntax

```
Public NotInheritable Class DateTimeWithTimeZone
```

C# syntax

```
public sealed class DateTimeWithTimeZone
```

Members

All members of DateTimeWithTimeZone class, including all inherited members.

Name	Description
"DateTimeWithTimeZone constructor"	Constructs a DateTimeWithTimeZone with the specified year, month, day, hour, minute, second, millisecond, time zone hour and time zone minute.
"Parse method"	Parses the given string and returns a new DateTimeWithTimeZone.
"ToString method"	The equivalent to DateTime.ToString() with the time zone offset appended to the end.
"DateTime property"	Gets the DateTime that corresponds to this without the time zone offset.
"Day property"	The equivalent to DateTime.Day.

Name	Description
"Hour property"	The equivalent to DateTime.Hour.
"Millisecond property"	The equivalent to DateTime.Millisecond.
"Minute property"	The equivalent to DateTime.Minute.
"Month property"	The equivalent to DateTime.Year.
"Second property"	The equivalent to DateTime.Second.
"TimeZoneHour property"	Gets the hours part of the time zone offset.
"TimeZoneMinute property"	Gets the minutes part of the time zone offset.
"Year property"	The equivalent to DateTime.Year.

DateTimeWithTimeZone constructor

Constructs a DateTimeWithTimeZone with the specified year, month, day, hour, minute, second, millisecond, time zone hour and time zone minute.

Overload list

Name	Description
"DateTimeWithTimeZone(DateTime) constructor"	Constructs a DateTimeWithTimeZone with the same date and time as the specified DateTime with time zone offsets of 0.
"DateTimeWithTimeZone(DateTime, int, int) constructor"	Constructs a DateTimeWithTimeZone with the same date and time as the specified DateTime with the provided time zone offset.
"DateTimeWithTimeZone(int, int, int, int, int, int, int, int) constructor"	Constructs a DateTimeWithTimeZone with the specified year, month, day, hour, minute, second, millisecond, time zone hour and time zone minute.

DateTimeWithTimeZone(DateTime) constructor

Constructs a DateTimeWithTimeZone with the same date and time as the specified DateTime with time zone offsets of 0.

Visual Basic syntax

```
Public Sub New(ByVal dt As Date)
```

C# syntax

```
public DateTimeWithTimeZone(DateTime dt)
```

DateTimeWithTimeZone(DateTime, int, int) constructor

Constructs a DateTimeWithTimeZone with the same date and time as the specified DateTime with the provided time zone offset.

Visual Basic syntax

```
Public Sub New(
    ByVal dt As Date,
    ByVal tz_hour As Integer,
    ByVal tz_minute As Integer
)
```

C# syntax

```
public DateTimeWithTimeZone(DateTime dt, int tz_hour, int tz_minute)
```

Parameters

- **dt** The date and time portions.
- **tz_hour** The time zone offset hours. (-12 through 14)
- **tz_minute** The time zone offset minutes. (-59 through 59). This value can be negative only if *tz_hour* is not positive.

Exceptions

- **System.ArgumentOutOfRangeException** Thrown when tz_hour or tz_minute are out of range.

DateTimeWithTimeZone(int, int, int, int, int, int, int, int, int) constructor

Constructs a DateTimeWithTimeZone with the specified year, month, day, hour, minute, second, millisecond, time zone hour and time zone minute.

Visual Basic syntax

```
Public Sub New(
    ByVal year As Integer,
    ByVal month As Integer,
    ByVal day As Integer,
    ByVal hour As Integer,
    ByVal minute As Integer,
    ByVal second As Integer,
    ByVal millisecond As Integer,
    ByVal tz_hour As Integer,
    ByVal tz_minute As Integer
)
```


C# syntax

```
public DateTimeWithTimeZone(
    int year,
    int month,
    int day,
    int hour,
    int minute,
    int second,
    int millisecond,
    int tz_hour,
    int tz_minute
)
```

Parameters

- **year** The year. (1 through 9999)
- **month** The month. (1 through 12)
- **day** The day. (1 through the number of days in *month*)
- **hour** The hours. (0 through 23)
- **minute** The minutes. (0 through 59)
- **second** The seconds. (0 through 59)
- **millisecond** The milliseconds. (1 through 999)
- **tz_hour** The time zone offset hours. (-12 through 14)
- **tz_minute** The time zone offset minutes. (-59 through 59). This value can be negative only if *tz_hour* is not positive.

Exceptions

- **System.ArgumentOutOfRangeException** Thrown when any parameter is out of range.

Parse method

Parses the given string and returns a new DateTimeWithTimeZone.

Visual Basic syntax

```
Public Shared Function Parse(
    ByVal val As String
) As DateTimeWithTimeZone
```

C# syntax

```
public static DateTimeWithTimeZone Parse(string val)
```

Parameters

- **val** A string of the form "yyyy-MM-dd HH:mm:ss.ffffff SHH:mm", where S is the sign on the time zone hours offset and the second HH:mm is the time zone offset. The fractional part of the time can be omitted. The time zone offset can be omitted. If the time zone offset is present, the sign can be omitted.

Exceptions

- **System.FormatException** Thrown when the given string does not match that format.

ToString method

The equivalent to DateTime.ToString() with the time zone offset appended to the end.

Overload list

Name	Description
"ToString() method"	The equivalent to DateTime.ToString() with the time zone offset appended to the end.
"ToString(IFormatProvider) method"	The equivalent to DateTime.ToString(provider) with the time zone offset appended to the end.
"ToString(string) method"	The equivalent to DateTime.ToString(format) with the time zone offset appended to the end.
"ToString(string, IFormatProvider) method"	The equivalent to DateTime.ToString(format, provider) with the time zone offset appended to the end.

ToString() method

The equivalent to DateTime.ToString() with the time zone offset appended to the end.

Visual Basic syntax

```
Public Overrides Function ToString() As String
```

C# syntax

```
public override string ToString()
```

ToString(IFormatProvider) method

The equivalent to DateTime.ToString(provider) with the time zone offset appended to the end.

Visual Basic syntax

```
Public Function ToString(ByVal provider As IFormatProvider) As String
```


C# syntax

```
public string ToString(IFormatProvider provider)
```

ToString(string) method

The equivalent to DateTime.ToString(format) with the time zone offset appended to the end.

Visual Basic syntax

```
Public Function ToString(ByVal format As String) As String
```

C# syntax

```
public string ToString(string format)
```

ToString(string, IFormatProvider) method

The equivalent to DateTime.ToString(format, provider) with the time zone offset appended to the end.

Visual Basic syntax

```
Public Function ToString(
    ByVal format As String,
    ByVal provider As IFormatProvider
) As String
```

C# syntax

```
public string ToString(string format, IFormatProvider provider)
```

DateTime property

Gets the DateTime that corresponds to this without the time zone offset.

Visual Basic syntax

```
Public Property DateTime As Date
```

C# syntax

```
public DateTime DateTime {get;set;}
```

Day property

The equivalent to DateTime.Day.

Visual Basic syntax

```
Public ReadOnly Property Day As Integer
```

C# syntax

```
public int Day {get;}
```

Hour property

The equivalent to DateTime.Hour.

Visual Basic syntax

```
Public ReadOnly Property Hour As Integer
```

C# syntax

```
public int Hour {get;}
```

Millisecond property

The equivalent to DateTime.Millisecond.

Visual Basic syntax

```
Public ReadOnly Property Millisecond As Integer
```

C# syntax

```
public int Millisecond {get;}
```

Minute property

The equivalent to DateTime.Minute.

Visual Basic syntax

```
Public ReadOnly Property Minute As Integer
```

C# syntax

```
public int Minute {get;}
```

Month property

The equivalent to DateTime.Year.

Visual Basic syntax

```
Public ReadOnly Property Month As Integer
```


C# syntax

```
public int Month {get;}
```

Second property

The equivalent to DateTime.Second.

Visual Basic syntax

```
Public ReadOnly Property Second As Integer
```

C# syntax

```
public int Second {get;}
```

TimeZoneHour property

Gets the hours part of the time zone offset.

Visual Basic syntax

```
Public Property TimeZoneHour As Integer
```

C# syntax

```
public int TimeZoneHour {get;set;}
```

TimeZoneMinute property

Gets the minutes part of the time zone offset.

Visual Basic syntax

```
Public Property TimeZoneMinute As Integer
```

C# syntax

```
public int TimeZoneMinute {get;set;}
```

Year property

The equivalent to DateTime.Year.

Visual Basic syntax

```
Public ReadOnly Property Year As Integer
```


C# syntax

```
public int Year {get;}
```

DBCommand interface

Represents a SQL statement or database command.

Visual Basic syntax

```
Public Interface DBCommand
```

C# syntax

```
public interface DBCommand
```

Members

All members of DBCommand interface, including all inherited members.

Name	Description
"Close method"	Closes the current SQL statement or command.
"ExecuteNonQuery method"	Executes a non-query statement.
"ExecuteReader method"	Executes a query statement returning the result set.
"Prepare method"	Prepares the SQL statement stored in CommandText for execution.
"CommandText property"	The SQL statement to be executed.
"Parameters property"	Gets the DBParameterCollection for this DBCommand.

Remarks

DBCommand can represent an update or query.

Example

The following C# code uses the DBCommand interface to execute two queries:

```
DBCommand stmt = conn.CreateCommand();
stmt.CommandText = "SELECT t1a1, t1a2 FROM table1 ";

DBRowReader rs = stmt.ExecuteReader();
printResultSet( rs );
rs.Close();

stmt.CommandText = "SELECT t2a1 FRONT table2 ";
rs = stmt.ExecuteReader();
printResultSet( rs );
rs.Close();
stmt.Close();
```


The following C# code uses the DBCommand interface to execute an update with parameters:

```
public void prepare_for_download(DateTime last_download,
    String ml_username)
{
        DBCommand cstmt = conn.CreateCommand();
        cstmt.CommandText = "CALL myProc( ?,?,? )";
        cstmt.Prepare();

        DBParameter param = new DBParameter();
        param.DbType          = SQLType.SQL_CHAR;
        param.Value           = "10000";
        cstmt.Parameters.Add(param);

        param                 = new DBParameter();
        param.DbType          = SQLType.SQL_INTEGER;
        param.Value           = 20000;
        cstmt.Parameters.Add(param);

        param                 = new DBParameter();
        param.DbType          = SQLType.SQL_DECIMAL;
        param.Precision       = 5;
        param.Value           = new Decimal(30000);
        cstmt.Parameters.Add(param);

        // Execute update
        DBRowReader rset = cstmt.ExecuteNonQuery();
        cstmt.Close();
    }
```

Close method

Closes the current SQL statement or command.

Visual Basic syntax

```
Public Sub Close()
```

C# syntax

```
public void Close()
```

ExecuteNonQuery method

Executes a non-query statement.

Visual Basic syntax

```
Public Function ExecuteNonQuery() As Integer
```

C# syntax

```
public int ExecuteNonQuery()
```

Returns

The number of rows in the database affected by the SQL statement.

ExecuteReader method

Executes a query statement returning the result set.

Visual Basic syntax

```
Public Function ExecuteReader() As DBRowReader
```

C# syntax

```
public DBRowReader ExecuteReader()
```

Returns

A DBRowReader for retrieving results returned by the SQL statement.

Prepare method

Prepares the SQL statement stored in CommandText for execution.

Visual Basic syntax

```
Public Sub Prepare()
```

C# syntax

```
public void Prepare()
```

CommandText property

The SQL statement to be executed.

Visual Basic syntax

```
Public Property CommandText As String
```

C# syntax

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

Parameters property

Gets the DBParameterCollection for this DBCommand.

Visual Basic syntax

```
Public ReadOnly Property Parameters As DBParameterCollection
```

C# syntax

```
public DBParameterCollection Parameters {get;}
```

Returns

The requested Parameter collection.

See also

- "DBParameterCollection class" on page 624

DBConnection interface

Represents a MobiLink ODBC connection.

Visual Basic syntax

```
Public Interface DBConnection
```

C# syntax

```
public interface DBConnection
```

Members

All members of DBConnection interface, including all inherited members.

Name	Description
"Close method"	Closes the current connection.
"Commit method"	Commits the current transaction.
"CreateCommand method"	Creates a SQL statement or command on this connection.
"Rollback method"	Rolls back the current transaction.

Remarks

This interface allows user-written synchronization logic to access an ODBC connection created by MobiLink.

Close method

Closes the current connection.

Visual Basic syntax

```
Public Sub Close()
```

C# syntax

```
public void Close()
```

Commit method

Commits the current transaction.

Visual Basic syntax

```
Public Sub Commit()
```

C# syntax

```
public void Commit()
```

CreateCommand method

Creates a SQL statement or command on this connection.

Visual Basic syntax

```
Public Function CreateCommand() As DBCommand
```

C# syntax

```
public DBCommand CreateCommand()
```

Returns

The newly generated DBCommand.

Rollback method

Rolls back the current transaction.

Visual Basic syntax

```
Public Sub Rollback()
```

C# syntax

```
public void Rollback()
```

DBConnectionContext interface

Obtains information about the current database connection.

Visual Basic syntax

```
Public Interface DBConnectionContext
```

C# syntax

```
public interface DBConnectionContext
```

Members

All members of DBConnectionContext interface, including all inherited members.

Name	Description
"GetConnection method"	Returns the existing connection to the MobiLink consolidated database.
"GetDownloadData method"	Returns the DownloadData for the current synchronization.
"GetProperties method"	Returns a collection of properties based on this connections script version.
"GetRemoteID method"	Returns the remote ID of the database currently synchronizing on this connection.
"GetServerContext method"	Returns the current server context.
"GetVersion method"	Returns the version string for this connection.

Remarks

This is passed to the constructor of classes containing scripts. If context is required for a background thread or beyond the lifetime of a connection, use a ServerContext.

For more information about constructors, see "Constructors" on page 592.

Note
A DBConnectionContext instance should not be used outside the thread that calls into your .NET code.

Example

The following example shows you how to create a class level DBConnectionContext instance to use in your synchronization scripts. The DBConnectionContext getConnection method obtains a DBConnection instance representing the current connection with the MobiLink consolidated database.

```
using iAnywhere.MobiLink.Script;
using System.Data;

public class OrderProcessor {
    DBConnectionContext _cc;

    public OrderProcessor( DBConnectionContext cc ) {
    _cc = cc;
    }
```

```
        // The method used for the handle_DownloadData event.
        public void HandleEvent() {
            DBConnection my_connection = _cc.GetConnection();
            // ...
        }

        // ...
    }
```

GetConnection method

Returns the existing connection to the MobiLink consolidated database.

Visual Basic syntax

```
Public Function GetConnection() As DBConnection
```

C# syntax

```
public DBConnection GetConnection()
```

Returns

The current connection to the consolidated database. This connection is only valid for the lifetime of the underlying MobiLink connection.

Remarks

This is the same connection that MobiLink uses to execute SQL scripts. It must not be committed, closed, or altered in any way that would affect the MobiLink server use of the connection.

Do not use the connection after the end_connection event has been called for the connection.

Use the MakeConnection method if a server connection with full access is required.

See also

- "MakeConnection method" on page 651

GetDownloadData method

Returns the DownloadData for the current synchronization.

Visual Basic syntax

```
Public Function GetDownloadData() As DownloadData
```

C# syntax

```
public DownloadData GetDownloadData()
```

Returns

The DownloadData for the current synchronization; otherwise, null if the synchronization is upload-only.

Remarks

Use the DownloadData instance to create the download for direct row handling.

Example

The following example assumes you have created a DBConnectionContext instance named _cc:

```
// The method used for the handle_DownloadData event.
public void HandleDownload() {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.GetDownloadData();

    // Get an array of tables to set download operations.
        DownloadTableData[] download_tables = my_dd.GetDownloadTables();

        // Get the first table in the DownloadTableData array.
    DownloadTableData my_download_table = download_tables[0];

    // ...
}
```

GetProperties method

Returns a collection of properties based on this connections script version.

Visual Basic syntax

```
Public Function GetProperties() As NameValueCollection
```

C# syntax

```
public NameValueCollection GetProperties()
```

Returns

The properties for the current script version.

Remarks

Properties are stored in the ml_property table. For more information, see “ml_add_property system procedure” on page 705.

Example

The following example shows you how to output the properties for a DBConnectionContext.

> **Note**
> This example assumes you have a DBConnectionContext instance called _cc.

```
// The method used to output the connection properties.
public void OutputProperties() {
    // output the Properties for the current synchronization
    NameValueCollection properties = _cc.GetProperties();
    System.Console.WriteLine(properties.ToString());
}
```

GetRemoteID method

Returns the remote ID of the database currently synchronizing on this connection.

Visual Basic syntax

```
Public Function GetRemoteID() As String
```

C# syntax

```
public string GetRemoteID()
```

Returns

The remote ID.

Remarks

For more information about Remote IDs, see "Remote IDs" [*MobiLink - Client Administration*].

Example

The following example shows you how to output the remote ID for a DBConnectionContext.

```
// The method used to output the remote ID.
public void OutputRemoteID() {
    // output the Remote ID for the current synchronization
    string remoteID = _cc.GetRemoteID();
    System.Console.WriteLine(remoteID);
}
```

GetServerContext method

Returns the current server context.

Visual Basic syntax

```
Public Function GetServerContext() As ServerContext
```

C# syntax

```
public ServerContext GetServerContext()
```

Returns

The ServerContext for this MobiLink server.

Remarks

This method can be used to create new connections or interact with boot classes.

Example

The following example shows you how to get the ServerContext instance for a DBConnectionContext and shut down the server.


```
// A method that uses an instance of the ServerContext to shut down the
server
public void ShutDownServer() {
    ServerContext context = _cc.GetServerContext();
    context.Shutdown();
}
```

GetVersion method

Returns the version string for this connection.

Visual Basic syntax

```
Public Function GetVersion() As String
```

C# syntax

```
public string GetVersion()
```

Returns

The script version name.

Remarks

Properties are stored in the ml_property table. For more information, “ml_add_property system procedure” on page 705.

Example

The following example shows you how to get the script version and use it to make decisions.

```
public void MyEvent() {
    // ...

    string version = _cc.GetVersion();
    switch( version ) {
    case "My Version 1":
        // ...
        break;
    case "My Version 2":
        // ...
        break;
    }
}
```

DBParameter class

Represents a bound ODBC parameter.

Visual Basic syntax

```
Public Class DBParameter
```

C# syntax

```
public class DBParameter
```

Members

All members of DBParameter class, including all inherited members.

Name	Description
"DbType property"	The SQLType of this parameter.
"Direction property"	The Input/Output direction of this parameter.
"IsNullable property"	True if the parameter can be null; otherwise, false.
"ParameterName property"	The name of this parameter.
"Precision property"	The Decimal precision of this parameter.
"Scale property"	The resolvable digits of this parameter.
"Size property"	The size of this parameter, measured in bytes.
"Value property"	The value of this parameter.
"HasChanged field"	Returns whether the parameter has been modified since creation.

Remarks

This class is required to execute commands with parameters. All parameters must be in place before the command is executed.

Example

The following C# code uses the DBCommand interface to execute an update with parameters:

```
using( DBCommand cstmt = conn.CreateCommand() ) {

    DBCommand cstmt = conn.CreateCommand();

    cstmt.CommandText = "call myProc( ?,?,? )";

    cstmt.Prepare();

    DBParameter param = new DBParameter();
    param.DbType          = SQLType.SQL_CHAR;
    param.Value           = "10000";
    cstmt.Parameters.Add( param );

    param                 = new DBParameter();
    param.DbType          = SQLType.SQL_INTEGER;
    param.Value           = 20000;
    cstmt.Parameters.Add( param );

    param                 = new DBParameter();
    param.DbType          = SQLType.SQL_DECIMAL;
```



```
        param.Precision       = 5;
        param.Value           = new Decimal( 30000 );
        cstmt.Parameters.Add( param );

        // Execute update
        DBRowReader rset = cstmt.ExecuteNonQuery();
        cstmt.ExecuteNonQuery();
    }
```

DbType property

The SQLType of this parameter.

Visual Basic syntax

```
Public Property DbType As SQLType
```

C# syntax

```
public SQLType DbType {get;set;}
```

Remarks

The default value is SQLType.SQL_TYPE_NULL.

Direction property

The Input/Output direction of this parameter.

Visual Basic syntax

```
Public Property Direction As ParameterDirection
```

C# syntax

```
public ParameterDirection Direction {get;set;}
```

Remarks

The default value is ParameterDirection.Input.

IsNullable property

True if the parameter can be null; otherwise, false.

Visual Basic syntax

```
Public Property IsNullable As Boolean
```

C# syntax

```
public bool IsNullable {get;set;}
```

Remarks

The default value is false.

ParameterName property

The name of this parameter.

Visual Basic syntax

```
Public Property ParameterName As String
```

C# syntax

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

Remarks

The default value is null.

Precision property

The Decimal precision of this parameter.

Visual Basic syntax

```
Public Property Precision As UInteger
```

C# syntax

```
public uint Precision {get;set;}
```

Remarks

This property is only used for SQLType.SQL_NUMERIC and SQLType.SQL_DECIMAL parameters.

The default value is 0.

Scale property

The resolvable digits of this parameter.

Visual Basic syntax

```
Public Property Scale As Short
```

C# syntax

```
public short Scale {get;set;}
```


Remarks

This property is only used for SQLType.SQL_NUMERIC and SQLType.SQL_DECIMAL parameters.

The default value is 0.

Size property

The size of this parameter, measured in bytes.

Visual Basic syntax

```
Public Property Size As UInteger
```

C# syntax

```
public uint Size {get;set;}
```

Remarks

The default value is inferred from DbType.

Value property

The value of this parameter.

Visual Basic syntax

```
Public Property Value As Object
```

C# syntax

```
public object Value {get;set;}
```

Remarks

The default value is null.

HasChanged field

Returns whether the parameter has been modified since creation.

Visual Basic syntax

```
Public HasChanged As Boolean
```

C# syntax

```
public bool HasChanged;
```

DBParameterCollection class

Collection of DBParameters.

Visual Basic syntax

```
Public Class DBParameterCollection
    Implements System.Data.IDataParameterCollection
    Implements System.Collections.IList
    Implements System.Collections.ICollection
    Implements System.Collections.IEnumerable
```

C# syntax

```
public class DBParameterCollection :
    System.Data.IDataParameterCollection,
    System.Collections.IList,
    System.Collections.ICollection,
    System.Collections.IEnumerable
```

Base classes

- System.Data.IDataParameterCollection
- System.Collections.IList
- System.Collections.ICollection
- System.Collections.IEnumerable

Members

All members of DBParameterCollection class, including all inherited members.

Name	Description
"DBParameterCollection constructor"	Creates an empty list of DBParameters.
"Add method"	Adds the given parameter to the collection.
"Clear method"	Removes all parameters from the collection.
"Contains method"	Checks whether the collection contains a parameter with the specified name.
"CopyTo method"	Copies the contents of the collection into the given array starting at the specified index.
"GetEnumerator method"	Returns an enumerator for the collection.
"IndexOf method"	Returns the index of the parameter with the given name in the collection.
"Insert method"	Inserts the given DBParameter into the collection at the specified index.

Name	Description
"Remove method"	Removes the given DBParameter from the collection.
"RemoveAt method"	Removes the parameter with the given name from the collection.
"Count property"	The number of parameters in the collection.
"IsFixedSize property"	Returns false.
"IsReadOnly property"	Returns false.
"IsSynchronized property"	Returns false.
"SyncRoot property"	Used to synchronize access to the DBParameterCollection.
"this property"	Gets or sets the DBParameter with the given name in the collection.

Remarks

A DBParamterCollection is initially empty when created by DBCommand, and must be filled with appropriate parameters before the DBCommand executes.

DBParameterCollection constructor

Creates an empty list of DBParameters.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public DBParameterCollection()
```

Add method

Adds the given parameter to the collection.

Visual Basic syntax

```
Public Function Add(ByVal value As Object) As Integer
```

C# syntax

```
public int Add(object value)
```

Parameters

- **value** The DBParameter object to add to the collection.

Returns

The index of the added parameter in the collection.

See also

- "DBParameter class" on page 619

Clear method

Removes all parameters from the collection.

Visual Basic syntax

```
Public Sub Clear()
```

C# syntax

```
public void Clear()
```

Contains method

Checks whether the collection contains a parameter with the specified name.

Overload list

Name	Description
"Contains(object) method"	Returns true if the collection contains the given DBParameter.
"Contains(string) method"	Checks whether the collection contains a parameter with the specified name.

Contains(object) method

Returns true if the collection contains the given DBParameter.

Visual Basic syntax

```
Public Function Contains(ByVal value As Object) As Boolean
```

C# syntax

```
public bool Contains(object value)
```

Parameters

- **value** The DBParameter object to check for.

Returns

True if this collection contains the DBParameter; otherwise, false.

See also

- "DBParameter class" on page 619

Contains(string) method

Checks whether the collection contains a parameter with the specified name.

Visual Basic syntax

```
Public Function Contains(ByVal parameterName As String) As Boolean
```

C# syntax

```
public bool Contains(string parameterName)
```

Parameters

- **parameterName** The name of the parameter to check for.

Returns

True if this collection contains a parameter with the given name; otherwise, false.

CopyTo method

Copies the contents of the collection into the given array starting at the specified index.

Visual Basic syntax

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

C# syntax

```
public void CopyTo(Array array, int index)
```

Parameters

- **array** The array to which the contents of the collection are copied into.
- **index** The index in the array at which the contents of the array should be copied into the collection.

GetEnumerator method

Returns an enumerator for the collection.

Visual Basic syntax

```
Public Function GetEnumerator() As System.Collections.IEnumerator
```

C# syntax

```
public System.Collections.IEnumerator GetEnumerator()
```

Returns

An enumerator for the collection.

IndexOf method

Returns the index of the parameter with the given name in the collection.

Overload list

Name	Description
"IndexOf(object) method"	Returns the index of the given DBParameter in the collection.
"IndexOf(string) method"	Returns the index of the parameter with the given name in the collection.

IndexOf(object) method

Returns the index of the given DBParameter in the collection.

Visual Basic syntax

```
Public Function IndexOf(ByVal value As Object) As Integer
```

C# syntax

```
public int IndexOf(object value)
```

Parameters

- **value** The DBParameter object to find.

Returns

The index of the DBParameter in the collection.

See also

- "DBParameter class" on page 619

IndexOf(string) method

Returns the index of the parameter with the given name in the collection.

Visual Basic syntax

```
Public Function IndexOf(ByVal parameterName As String) As Integer
```


C# syntax

```
public int IndexOf(string parameterName)
```

Parameters

- **parameterName** The name of the parameter to find.

Returns

The index of the parameter, or -1 if there is no parameter with the given name.

Insert method

Inserts the given DBParameter into the collection at the specified index.

Visual Basic syntax

```
Public Sub Insert(ByVal index As Integer, ByVal value As Object)
```

C# syntax

```
public void Insert(int index, object value)
```

Parameters

- **value** The DBParameter object to insert.
- **index** The index at which to insert the value.

See also

- "DBParameter class" on page 619

Remove method

Removes the given DBParameter from the collection.

Visual Basic syntax

```
Public Sub Remove(ByVal value As Object)
```

C# syntax

```
public void Remove(object value)
```

Parameters

- **value** The DBParameter to remove.

See also

- "DBParameter class" on page 619

RemoveAt method

Removes the parameter with the given name from the collection.

Overload list

Name	Description
"RemoveAt(int) method"	Removes the DBParameter at the given index in the collection.
"RemoveAt(string) method"	Removes the parameter with the given name from the collection.

RemoveAt(int) method

Removes the DBParameter at the given index in the collection.

Visual Basic syntax

```
Public Sub RemoveAt(ByVal index As Integer)
```

C# syntax

```
public void RemoveAt(int index)
```

Parameters

- **index** The index of the DBParameter to remove.

See also

- "DBParameter class" on page 619

RemoveAt(string) method

Removes the parameter with the given name from the collection.

Visual Basic syntax

```
Public Sub RemoveAt(ByVal parameterName As String)
```

C# syntax

```
public void RemoveAt(string parameterName)
```

Parameters

- **parameterName** The name of the parameter to remove.

Count property

The number of parameters in the collection.

Visual Basic syntax

```
Public ReadOnly Property Count As Integer
```

C# syntax

```
public int Count {get;}
```

IsFixedSize property

Returns false.

Visual Basic syntax

```
Public ReadOnly Property IsFixedSize As Boolean
```

C# syntax

```
public bool IsFixedSize {get;}
```

IsReadOnly property

Returns false.

Visual Basic syntax

```
Public ReadOnly Property IsReadOnly As Boolean
```

C# syntax

```
public bool IsReadOnly {get;}
```

IsSynchronized property

Returns false.

Visual Basic syntax

```
Public ReadOnly Property IsSynchronized As Boolean
```

C# syntax

```
public bool IsSynchronized {get;}
```

SyncRoot property

Used to synchronize access to the DBParameterCollection.

Visual Basic syntax

```
Public ReadOnly Property SyncRoot As Object
```

C# syntax

```
public object SyncRoot {get;}
```

this property

Gets or sets the DBParameter with the given name in the collection.

Overload list

Name	Description
"this[int] property"	Gets or sets the DBParameter at the given index in the collection.
"this[string] property"	Gets or sets the DBParameter with the given name in the collection.

this[int] property

Gets or sets the DBParameter at the given index in the collection.

Visual Basic syntax

```
Public Property Item(ByVal index As Integer) As Object
```

C# syntax

```
public object this[int index] {get;set;}
```

Parameters

- **index** The index of the DBParameter to get or set.

Returns

This with the given index in the collection.

See also

- "DBParameter class" on page 619

this[string] property

Gets or sets the DBParameter with the given name in the collection.

Visual Basic syntax

```
Public Property Item(ByVal parameterName As String) As Object
```


C# syntax

```
public object this[string parameterName] {get;set;}
```

Parameters

- **parameterName** The name of the DBParameter to get or set.

Returns

This with the given name in the collection.

See also

- "DBParameter class" on page 619

DBRowReader interface

Represents a set of rows being read from a database.

Visual Basic syntax

```
Public Interface DBRowReader
```

C# syntax

```
public interface DBRowReader
```

Members

All members of DBRowReader interface, including all inherited members.

Name	Description
"Close method"	Cleans up all resources used by this MLDBRowReader.
"NextRow method"	Retrieves and returns the next row in the result set.
"ColumnNames property"	Gets the names of all columns in the result set.
"ColumnTypes property"	Gets the types of all columns in the result set.

Remarks

Executing the ExecuteReader method creates a DBRowReader.

See also

- "ExecuteReader method" on page 612

Example

The following C# code calls a function with the rows in the result set represented by the given DBRowReader:

```
        DBCommand stmt = conn.CreateCommand();
        stmt.CommandText = "select intCol, strCol from table1 ";
        DBRowReader rs = stmt.ExecuteReader();
    object[] values = rset.NextRow();

        while( values != null ) {
            handleRow( (int)values[0], (String)values[1] );
            values = rset.NextRow();
        }
        rset.Close();
        stmt.Close();
```

Close method

Cleans up all resources used by this MLDBRowReader.

Visual Basic syntax

```
Public Sub Close()
```

C# syntax

```
public void Close()
```

Remarks

This MLDBRowReader cannot be used again after this method is called.

NextRow method

Retrieves and returns the next row in the result set.

Visual Basic syntax

```
Public Function NextRow() As Object()
```

C# syntax

```
public object[] NextRow()
```

Returns

The next row of values in the result set, or null if there are no more rows in this result set.

See also

- "SQLType enumeration" on page 668

ColumnNames property

Gets the names of all columns in the result set.

Visual Basic syntax

```
Public ReadOnly Property ColumnNames As String()
```

C# syntax

```
public string[] ColumnNames {get;}
```

Remarks

The value is an array of strings corresponding to the column names in the result set.

ColumnTypes property

Gets the types of all columns in the result set.

Visual Basic syntax

```
Public ReadOnly Property ColumnTypes As SQLType()
```

C# syntax

```
public SQLType[] ColumnTypes {get;}
```

Remarks

The value is an array of SQLTypes corresponding to the column types in the result set.

DownloadData interface

Encapsulates all download data operations for direct row handling.

Visual Basic syntax

```
Public Interface DownloadData
```

C# syntax

```
public interface DownloadData
```

Members

All members of DownloadData interface, including all inherited members.

Name	Description
"GetDownloadTableByName method"	Gets the named download table for this synchronization.
"GetDownloadTables method"	Gets an array of all the tables for download in this synchronization.

Remarks

Use the GetDownloadData method To obtain a DownloadData instance. Use the GetDownloadTables and GetDownloadTableByName methods to return DownloadTableData instances.

This download data is available through DBConnectionContext. It is not valid to access the download data before the begin_sync event, or the DownloadData in an upload only synchronization.

For more information about direct row handling, see "handle_DownloadData connection event" on page 383 and "Direct row handling" on page 671.

See also

- "DownloadTableData interface" on page 637
- "GetDownloadData method" on page 616

GetDownloadTableByName method

Gets the named download table for this synchronization.

Visual Basic syntax

```
Public Function GetDownloadTableByName(
    ByVal table_name As String
) As DownloadTableData
```

C# syntax

```
public DownloadTableData GetDownloadTableByName(string table_name)
```

Parameters

- **table_name** The name of the table for which you want the download data

Returns

The download data for the given table name, or null if not found.

GetDownloadTables method

Gets an array of all the tables for download in this synchronization.

Visual Basic syntax

```
Public Function GetDownloadTables() As DownloadTableData()
```

C# syntax

```
public DownloadTableData[] GetDownloadTables()
```


Returns

An array of download table data. The order of tables in the array is the same as the upload order for the remote.

Remarks

The operations performed on this table are sent to the remote database.

Example

The following example uses the GetDownloadTables method to obtain an array of DownloadTableData objects for the current synchronization. The example assumes you have a DBConnectionContext instance named _cc.

```
// The method used for the handle_DownloadData event.
public void HandleDownload() {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.GetDownloadData();

    // Get an array of tables to set download operations.
    DownloadTableData[] download_tables = my_dd.GetDownloadTables();

    // Get the first table in the DownloadTableData array.
    DownloadTableData my_download_table = download_tables[0];

    // ...
}
```

DownloadTableData interface

Encapsulates information for one download table for a synchronization.

Visual Basic syntax

```
Public Interface DownloadTableData
```

C# syntax

```
public interface DownloadTableData
```

Members

All members of DownloadTableData interface, including all inherited members.

Name	Description
"GetDeleteCommand method"	Get a command which allows the user to add delete operations to the download data operations.
"GetLastDownloadTime method"	Returns the last download time for this table.
"GetName method"	Gets the table name of this instance.

Name	Description
"GetSchemaTable method"	Gets a DataTable that describes the metadata for this download table.
"GetUpsertCommand method"	Get a command which allows the user to add upsert(insert/update) operations to the download data operations.

Remarks

Encapsulates table operations for MobiLink direct downloads. Use this interface to set the data operations that are downloaded to the client. To obtain DownloadTableData instances for the current synchronization, use the DownloadData interface. You can use the DownloadTableData.GetUpsertCommand and GetDeleteCommand methods to obtain IDbCommand objects for insert and update, and delete operations, respectively. The System.Data.IDbCommand.ExecuteNonQuery() method registers an operation for download.

Example

Suppose you have the following table:

```
CREATE TABLE remoteOrders (
    pk INT NOT NULL,
    col1 VARCHAR(200),
    PRIMARY KEY (pk)
);
```

The following example uses the GetDownloadTableByName method to return a DownloadTableData instance representing the remoteOrders table:

```
// The method used for the handle_DownloadData event
public void HandleDownload() {
    // _cc is a DBConnectionContext instance.

    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.GetDownloadData();

    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData td = my_dd.GetDownloadTableByName("remoteOrders");

    // User defined-methods to set download operations.
    SetDownloadUpserts(td);
    SetDownloadDeletes(td);

    // ...
}
```

In this example, the SetDownloadInserts method uses GetUpsertCommand to obtain a command for the rows you want to insert or update. The IDbCommand holds the parameters that you set to the values you want inserted on the remote database.

```
void SetDownloadInserts(DownloadTableData td) {
    IDbCommand  upsert_cmd = td.GetUpsertCommand();
    IDataParameterCollection parameters = upsert_cmd.Parameters;

    // The following method calls are the same as the following SQL
statement:
    // INSERT INTO remoteOrders(pk, col1) values(2300, "truck");
```



```
    ((IDataParameter) (parameters[0])).Value = (Int32) 2300;
    ((IDataParameter) (parameters[1])).Value = (String) "truck";

    if (upsert_cmd.ExecuteNonQuery() > 0) {
        // Insert was not filtered.
    }
    else {
        // Insert was filtered because it was uploaded
        // in the same synchronization.
    }
 }
```

The following method uses the DownloadTableData.GetDeleteCommand to obtain a command for rows you want to delete.

```
void SetDownloadDeletes(DownloadTableData td) {
    IDbCommand  delete_cmd = t2_download_dd.GetDeleteCommand();

    // The following method calls are the same as the following SQL
statement:
    // DELETE FROM remoteOrders  where pk = 2300;
    IDataParameterCollection parameters = delete_cmd.Parameters;
    ((IDataParameter) (parameters[0])).Value = (Int32) 2300;
    delete_cmd.ExecuteNonQuery();
}
```

GetDeleteCommand method

Get a command which allows the user to add delete operations to the download data operations.

Visual Basic syntax

```
Public Function GetDeleteCommand() As IDbCommand
```

C# syntax

```
public IDbCommand GetDeleteCommand()
```

Returns

A command for deletes in the download.

Remarks

The command returned has the same number of parameters as primary key columns in this table. The column values for the primary key columns must be set and the statement executed with ExecuteNonQuery for the delete to be included in the download. ExecuteNonQuery on the command returns 0 if the delete operation was filtered and returns 1 if the insert was not filtered.

To delete a row, you must set all primary key values for download delete operations. To truncate the remote table, set all primary key columns to null.

See also

- “DownloadTableData interface” on page 637

GetLastDownloadTime method

Returns the last download time for this table.

Visual Basic syntax

```
Public Function GetLastDownloadTime() As Date
```

C# syntax

```
public DateTime GetLastDownloadTime()
```

Returns

The last download time for this table.

Remarks

This is the same last download time passed to several of the per table download events.

The last download time is useful for generating the table download data for a particular synchronization.

GetName method

Gets the table name of this instance.

Visual Basic syntax

```
Public Function GetName() As String
```

C# syntax

```
public string GetName()
```

Returns

The table name of this instance.

Remarks

This is a utility function. The table name can also be accessed via the Schema for this instance.

GetSchemaTable method

Gets a DataTable that describes the metadata for this download table.

Visual Basic syntax

```
Public Function GetSchemaTable() As DataTable
```

C# syntax

```
public DataTable GetSchemaTable()
```


Returns

A DataTable that describes the column metadata.

Remarks

You must specify the client option to send column names if you want the DataTable to contain column name information. Send column names is specified by default.

GetUpsertCommand method

Get a command which allows the user to add upsert(insert/update) operations to the download data operations.

Visual Basic syntax

```
Public Function GetUpsertCommand() As IDbCommand
```

C# syntax

```
public IDbCommand GetUpsertCommand()
```

Returns

A command for inserts/updates for the download.

Remarks

The command returned has the same number of parameters as columns in this table. The column values for the insert must be set and the statement executed with ExecuteNonQuery for the insert/update to be included in the download. ExecuteNonQuery on the command returns 0 if the insert operation was filtered and returns 1 if the insert was not filtered.

You cannot add or remove parameters to this command; you can only set their values.

See also

- "DownloadTableData interface" on page 637

FatalException class

Signals MobiLink that a fatal server-side error has occurred internally and should shutdown immediately.

Visual Basic syntax

```
Public Class FatalException Inherits System.ApplicationException
```

C# syntax

```
public class FatalException : System.ApplicationException
```

Base classes

- System.ApplicationException

Members

All members of FatalException class, including all inherited members.

Name	Description
"FatalException constructor"	Creates a new FatalException with the default message.

FatalException constructor

Creates a new FatalException with the default message.

Overload list

Name	Description
"FatalException() constructor"	Creates a new FatalException with the default message.
"FatalException(string) constructor"	Creates a new FatalException with the given message.
"FatalException(string, Exception) constructor"	Creates a new FatalException with the given message and containing the given inner exception that caused this one.

FatalException() constructor

Creates a new FatalException with the default message.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public FatalException()
```

FatalException(string) constructor

Creates a new FatalException with the given message.

Visual Basic syntax

```
Public Sub New(ByVal message As String)
```


C# syntax

```
public FatalException(string message)
```

Parameters

- **message** The message for this FatalException.

FatalException(string, Exception) constructor

Creates a new FatalException with the given message and containing the given inner exception that caused this one.

Visual Basic syntax

```
Public Sub New(ByVal message As String, ByVal ie As Exception)
```

C# syntax

```
public FatalException(string message, Exception ie)
```

Parameters

- **message** The message for this FatalException.
- **ie** The exception that caused this FatalException.

LogMessage class

Contains information regarding a message printed to the log.

Visual Basic syntax

```
Public Class LogMessage
```

C# syntax

```
public class LogMessage
```

Members

All members of LogMessage class, including all inherited members.

Name	Description
"LogMessage constructor"	Create a LogMessage with the given attributes.
"MessageType enumeration"	Enumerates all possible LogMessage types.
"Text property"	The main text of the message.
"Type property"	The type of log message that this instance represents.

Name	Description
"User property"	The user for which this message is being logged.

Remarks

An instance of this class is passed into a LogCallback.

See also

- "LogCallback delegate" on page 667

LogMessage constructor

Create a LogMessage with the given attributes.

Visual Basic syntax

```
Public Sub New(
    ByVal type As MessageType,
    ByVal user As String,
    ByVal text As String
)
```

C# syntax

```
public LogMessage(MessageType type, string user, string text)
```

MessageType enumeration

Enumerates all possible LogMessage types.

Visual Basic syntax

```
Public Enum MessageType
```

C# syntax

```
public enum MessageType
```

Members

Member name	Description
ERROR	A log message is an error.
WARNING	A log message is a warning.
INFO	A log information message.

Text property

The main text of the message.

Visual Basic syntax

```
Public ReadOnly Property Text As String
```

C# syntax

```
public string Text {get;}
```

Type property

The type of log message that this instance represents.

Visual Basic syntax

```
Public ReadOnly Property Type As MessageType
```

C# syntax

```
public MessageType Type {get;}
```

User property

The user for which this message is being logged.

Visual Basic syntax

```
Public ReadOnly Property User As String
```

C# syntax

```
public string User {get;}
```

Remarks

This property can be null.

ScriptExecutionException class

Signals that an error has occurred in a user script.

Visual Basic syntax

```
Public Class ScriptExecutionException
    Inherits System.ApplicationException
```

C# syntax

```
public class ScriptExecutionException : System.ApplicationException
```

Base classes

- System.ApplicationException

Derived classes

- "ServerException class" on page 652
- "SynchronizationException class" on page 656

Members

All members of ScriptExecutionException class, including all inherited members.

Name	Description
"ScriptExecutionException constructor"	Creates a new ScriptExecutionException with the default message.

Remarks

Throwing this exception or any derivations of it, except for SychronizationException, causes the MobiLink server to shut down.

ScriptExecutionException constructor

Creates a new ScriptExecutionException with the default message.

Overload list

Name	Description
"ScriptExecutionException() constructor"	Creates a new ScriptExecutionException with the default message.
"ScriptExecutionException(string) constructor"	Creates a new ScriptExecutionException with the given message.
"ScriptExecutionException(string, Exception) constructor"	Creates a new ScriptExecutionException with the given message and containing the given inner exception that caused this one.

ScriptExecutionException() constructor

Creates a new ScriptExecutionException with the default message.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ScriptExecutionException()
```

ScriptExecutionException(string) constructor

Creates a new ScriptExecutionException with the given message.

Visual Basic syntax

```
Public Sub New(ByVal message As String)
```

C# syntax

```
public ScriptExecutionException(string message)
```

Parameters

- **message** The message for this ScriptExecutionException.

ScriptExecutionException(string, Exception) constructor

Creates a new ScriptExecutionException with the given message and containing the given inner exception that caused this one.

Visual Basic syntax

```
Public Sub New(ByVal message As String, ByVal ie As Exception)
```

C# syntax

```
public ScriptExecutionException(string message, Exception ie)
```

Parameters

- **message** The message for this ScriptExecutionException.
- **ie** The exception that caused this ScriptExecutionException.

ServerContext interface

Instantiates the context that is present for the duration of the MobiLink server.

Visual Basic syntax

```
Public Interface ServerContext
```

C# syntax

```
public interface ServerContext
```

Members

All members of ServerContext interface, including all inherited members.

Name	Description
"getProperties method"	Returns a collection of the properties for the given component and set.
"getPropertiesByVersion method"	Returns a collection of the properties for the given script version.
"getPropertySetNames method"	Returns a collection of the set names for the given component.
"GetStartClassInstances method"	Returns all start classes loaded into this domain.
"MakeConnection method"	Creates a new database connection.
"Shutdown method"	Causes the MobiLink server to perform a soft shutdown.
"ErrorListener event"	Triggered when the MobiLink server prints an error.
"InfoListener event"	Triggered when the MobiLink server prints information.
"ShutdownListener event"	Triggered when the MobiLink server is shutting down.
"WarningListener event"	Triggered when the MobiLink server prints an warning.

Remarks

This context can be held as static data and used in a background thread. It is valid for the duration of the .NET CLR invoked by MobiLink.

Use the GetServerContext method to access a ServerContext instance. It is passed to the constructor of boot classes.

See also

- "GetServerContext method" on page 618

getProperties method

Returns a collection of the properties for the given component and set.

Visual Basic syntax

```
Public Function getProperties(
    ByVal component As String,
    ByVal set As String
) As NameValueCollection
```


C# syntax

```
public NameValueCollection getProperties(string component, string set)
```

Parameters

- **component** Refers to the component name of the ml_property table.
- **set** Refers to the property set name of the ml_property table.

Returns

The properties for the given component/set.

Remarks

Properties are stored in the ml_property table. For more information, "ml_add_property system procedure" on page 705.

getPropertiesByVersion method

Returns a collection of the properties for the given script version.

Visual Basic syntax

```
Public Function getPropertiesByVersion(
    ByVal script_version As String
) As NameValueCollection
```

C# syntax

```
public NameValueCollection getPropertiesByVersion(string script_version)
```

Parameters

- **script_version** The script version for which to return associated properties.

Returns

The properties for the given script version.

Remarks

Properties are stored in the ml_property table. For more information, "ml_add_property system procedure" on page 705.

getPropertySetNames method

Returns a collection of the set names for the given component.

Visual Basic syntax

```
Public Function getPropertySetNames(
    ByVal component As String
) As StringCollection
```

C# syntax

```
public StringCollection getPropertySetNames(string component)
```

Parameters

- **component** Refers to the component name of the ml_property table.

Returns

The collection of set names for the given component.

Remarks

Properties are stored in the ml_property table. For more information, "ml_add_property system procedure" on page 705.

GetStartClassInstances method

Returns all start classes loaded into this domain.

Visual Basic syntax

```
Public Function GetStartClassInstances() As Object()
```

C# syntax

```
public object[] GetStartClassInstances()
```

Returns

An array of all start classes that were constructed at the server start time. The array length is zero if there are no start classes.

Remarks

For more information about user-defined start classes, see "User-defined start classes" on page 593.

Example

The following example demonstrates how to find a start class:

```
void FindStartClass( ServerContext sc, string name )
{
    object[] startClasses = sc.GetStartClassInstances();

    foreach( object obj in startClasses ) {
        if( obj is MyClass ) {
            // Execute some code.....
        }
    }
}
```


MakeConnection method

Creates a new database connection.

Visual Basic syntax

```
Public Function MakeConnection() As DBConnection
```

C# syntax

```
public DBConnection MakeConnection()
```

Returns

A new connection.

Shutdown method

Causes the MobiLink server to perform a soft shutdown.

Visual Basic syntax

```
Public Sub Shutdown()
```

C# syntax

```
public void Shutdown()
```

ErrorListener event

Triggered when the MobiLink server prints an error.

Visual Basic syntax

```
Public Event ErrorListener As LogCallback
```

C# syntax

```
public event LogCallback ErrorListener;
```

InfoListener event

Triggered when the MobiLink server prints information.

Visual Basic syntax

```
Public Event InfoListener As LogCallback
```

C# syntax

```
public event LogCallback InfoListener;
```

ShutdownListener event

Triggered when the MobiLink server is shutting down.

Visual Basic syntax

```
Public Event ShutdownListener As ShutdownCallback
```

C# syntax

```
public event ShutdownCallback ShutdownListener;
```

WarningListener event

Triggered when the MobiLink server prints an warning.

Visual Basic syntax

```
Public Event WarningListener As LogCallback
```

C# syntax

```
public event LogCallback WarningListener;
```

ServerException class

Sends a signal to MobiLink when an error has occurred with the server to indicate that it should shut down immediately.

Visual Basic syntax

```
Public Class ServerException Inherits ScriptExecutionException
```

C# syntax

```
public class ServerException : ScriptExecutionException
```

Base classes

- "ScriptExecutionException class" on page 645

Members

All members of ServerException class, including all inherited members.

Name	Description
"ServerException constructor"	Creates a new ServerException with the default message.
"ScriptExecutionException constructor"	Creates a new ScriptExecutionException with the default message.

ServerException constructor

Creates a new ServerException with the default message.

Overload list

Name	Description
"ServerException() constructor"	Creates a new ServerException with the default message.
"ServerException(string) constructor"	Creates a new ServerException with the given message.
"ServerException(string, Exception) constructor"	Creates a new ServerException with the given message and containing the given inner exception that caused this one.

ServerException() constructor

Creates a new ServerException with the default message.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public ServerException()
```

ServerException(string) constructor

Creates a new ServerException with the given message.

Visual Basic syntax

```
Public Sub New(ByVal message As String)
```

C# syntax

```
public ServerException(string message)
```

Parameters

- **message** The message for this ServerException.

ServerException(string, Exception) constructor

Creates a new ServerException with the given message and containing the given inner exception that caused this one.

Visual Basic syntax

```
Public Sub New(ByVal message As String, ByVal ie As Exception)
```

C# syntax

```
public ServerException(string message, Exception ie)
```

Parameters

- **message** The message for this ServerException.
- **ie** The exception that caused this ServerException.

SpatialUtilities class

Represents a collection of static methods to work with spatial values.

Visual Basic syntax

```
Public NotInheritable Class SpatialUtilities
```

C# syntax

```
public sealed class SpatialUtilities
```

Members

All members of SpatialUtilities class, including all inherited members.

Name	Description
"CreateSpatialValue method"	Returns a new byte array that contains a spatial value formatted for download: the first four bytes contain the given SRID in little endian, and the remainder is the spatial data passed in the given byte array.
"GetBytes method"	Returns a new byte array with the same spatial data as the given byte array, but with the SRID removed.
"GetSRID method"	Returns the SRID for the given spatial value.
"SetSRID method"	Stores the given SRID in the first four bytes of the given byte array.

CreateSpatialValue method

Returns a new byte array that contains a spatial value formatted for download: the first four bytes contain the given SRID in little endian, and the remainder is the spatial data passed in the given byte array.

Visual Basic syntax

```
Public Shared Function CreateSpatialValue(
    ByVal srid As Integer,
    ByVal spatial_value As Byte()
) As Byte()
```

C# syntax

```
public static byte[] CreateSpatialValue(int srid, byte[] spatial_value)
```

Parameters

- **srid** The SRID.
- **spatial_value** The spatial data.

Returns

The spatial value formatted for download.

GetBytes method

Returns a new byte array with the same spatial data as the given byte array, but with the SRID removed.

Visual Basic syntax

```
Public Shared Function GetBytes(ByVal spatial_value As Byte()) As Byte()
```

C# syntax

```
public static byte[] GetBytes(byte[] spatial_value)
```

Parameters

- **spatial_value** A spatial value that needs its SRID removed

Returns

The new byte array.

GetSRID method

Returns the SRID for the given spatial value.

Visual Basic syntax

```
Public Shared Function GetSRID(ByVal spatial_value As Byte()) As Integer
```

C# syntax

```
public static int GetSRID(byte[] spatial_value)
```

Parameters

- **spatial_value** The uploaded value. The first four bytes must contain the SRID encoded in little endian.

Returns

The SRID.

SetSRID method

Stores the given SRID in the first four bytes of the given byte array.

Visual Basic syntax

```
Public Shared Sub SetSRID(
    ByVal spatial_value As Byte(),
    ByVal srid As Integer
)
```

C# syntax

```
public static void SetSRID(byte[] spatial_value, int srid)
```

Parameters

- **spatial_value** The array to store the SRID in.
- **srid** The SRID to store

SynchronizationException class

Indicates when a synchronization exception has occurred and that the current synchronization should be rolled back and restarted.

Visual Basic syntax

```
Public Class SynchronizationException Inherits ScriptExecutionException
```

C# syntax

```
public class SynchronizationException : ScriptExecutionException
```

Base classes

- "ScriptExecutionException class" on page 645

Members

All members of SynchronizationException class, including all inherited members.

Name	Description
"SynchronizationException constructor"	Creates a new SynchronizationException with the default message.
"ScriptExecutionException constructor"	Creates a new ScriptExecutionException with the default message.

SynchronizationException constructor

Creates a new SynchronizationException with the default message.

Overload list

Name	Description
"SynchronizationException() constructor"	Creates a new SynchronizationException with the default message.
"SynchronizationException(string) constructor"	Creates a new SynchronizationException with the given message.
"SynchronizationException(string, Exception) constructor"	Creates a new SynchronizationException with the given message and containing the given inner exception that caused this one.

SynchronizationException() constructor

Creates a new SynchronizationException with the default message.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public SynchronizationException()
```

SynchronizationException(string) constructor

Creates a new SynchronizationException with the given message.

Visual Basic syntax

```
Public Sub New(ByVal message As String)
```

C# syntax

```
public SynchronizationException(string message)
```

Parameters

- **message** The message for this SynchronizationException.

SynchronizationException(string, Exception) constructor

Creates a new SynchronizationException with the given message and containing the given inner exception that caused this one.

Visual Basic syntax

```
Public Sub New(ByVal message As String, ByVal ie As Exception)
```

C# syntax

```
public SynchronizationException(string message, Exception ie)
```

Parameters

- **message** The message for this SynchronizationException.
- **ie** The exception that caused this SynchronizationException.

UpdateDataReader interface

Holds the update operations for one upload transaction for one table.

Visual Basic syntax

```
Public Interface UpdateDataReader Implements System.Data.IDataReader
```

C# syntax

```
public interface UpdateDataReader : System.Data.IDataReader
```

Base classes

- System.Data.IDataReader

Members

All members of UpdateDataReader interface, including all inherited members.

Name	Description
Close method (Inherited from System.Data.IDataReader)	Closes the System.Data.IDataReader Object.
GetSchemaTable method (Inherited from System.Data.IDataReader)	Returns a System.Data.DataTable that describes the column metadata of the System.Data.IDataReader.

Name	Description
NextResult method (Inherited from System.Data.IDataReader)	Advances the data reader to the next result, when reading the results of batch SQL statements.
Read method (Inherited from System.Data.IDataReader)	Advances the System.Data.IDataReader to the next record.
"SetNewRowValues method"	Sets the mode of this DataReader to return new column values (the post-updated row).
"SetOldRowValues method"	Sets the mode of this DataReader to return old column values (the pre-updated row).
Depth property (Inherited from System.Data.IDataReader)	Gets a value indicating the depth of nesting for the current row.
IsClosed property (Inherited from System.Data.IDataReader)	Gets a value indicating whether the data reader is closed.
RecordsAffected property (Inherited from System.Data.IDataReader)	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Remarks

New and old rows can be accessed by changing the mode of the DataReader to old or new. This interface can otherwise be used as a regular DataReader.

SetNewRowValues method

Sets the mode of this DataReader to return new column values (the post-updated row).

Visual Basic syntax

```
Public Sub SetNewRowValues()
```

C# syntax

```
public void SetNewRowValues()
```

Remarks

This is the default mode.

SetOldRowValues method

Sets the mode of this DataReader to return old column values (the pre-updated row).

Visual Basic syntax

```
Public Sub SetOldRowValues()
```

C# syntax

```
public void SetOldRowValues()
```

UploadData interface

Encapsulates upload operations for direct row handling.

Visual Basic syntax

```
Public Interface UploadData
```

C# syntax

```
public interface UploadData
```

Members

All members of UploadData interface, including all inherited members.

Name	Description
"GetUploadedTableByName method"	Gets the named Uploaded table data in this uploaded transaction.
"GetUploadedTables method"	Gets an array of all the uploaded tables data in this uploaded transaction.

Remarks

An upload transaction contains a set of tables containing row operations. An UploadData instance representing a single upload transaction is passed to the handle_UploadData synchronization event.

Note
You must handle direct row handling upload operations in the method registered for the handle_UploadData event. The UploadData is destroyed after each call to the registered method. Do not create a new instance of UploadData to use in subsequent events.

Use the UploadData.GetUploadedTables or UploadData.GetUploadedTableByName methods to obtain UploadedTableData instances.

A synchronization has one UploadData unless the remote database is using transactional or incremental upload.

Example

See "handle_UploadData connection event" on page 394.

GetUploadedTableByName method

Gets the named Uploaded table data in this uploaded transaction.

Visual Basic syntax

```
Public Function GetUploadedTableByName(
    ByVal table_name As String
) As UploadedTableData
```

C# syntax

```
public UploadedTableData GetUploadedTableByName(string table_name)
```

Parameters

- **table_name** The name of the table for which you want the uploaded data

Returns

The uploaded data for the given table name, or null if not found.

Example

Assuming that you use a method called HandleUpload for the handle_UploadData synchronization event, the following example uses the GetUploadedTableByName method to return an UploadedTableData instance for the remoteOrders table:

```
// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {
    UploadedTableData uploaded_t1 =
ut.GetUploadedTableByName("remoteOrders");
    // ...
}
```

GetUploadedTables method

Gets an array of all the uploaded tables data in this uploaded transaction.

Visual Basic syntax

```
Public Function GetUploadedTables() As UploadedTableData()
```

C# syntax

```
public UploadedTableData[] GetUploadedTables()
```

Returns

An array of uploaded table data. The order of tables in the array is the same as the upload order of the client.

Remarks

The order to the tables in the array is the same order that MobiLink uses for SQL row handling, and is the optimal order for preventing referential integrity violations. Use this table order if your data source is a relational database.

Example

Assuming that you use a method called HandleUpload for the handle_UploadData synchronization event, the following example uses the GetUploadedTables method to return UploadedTableData instances for the current upload transaction:

```
// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ud) {
    UploadedTableData[] tables = ud.GetUploadedTables();
    //...
}
```

UploadedTableData interface

Encapsulates information for one uploaded table for a synchronization.

Visual Basic syntax

```
Public Interface UploadedTableData
```

C# syntax

```
public interface UploadedTableData
```

Members

All members of UploadedTableData interface, including all inherited members.

Name	Description
"GetDeletes method"	Gets a DataReader with the deletes for this uploaded table data.
"GetInserts method"	Gets a DataReader with the inserts for this uploaded table data.
"GetName method"	Gets the table name of this instance.
"GetSchemaTable method"	Gets a DataTable that describes the metadata for this download table.
"GetUpdates method"	Gets a DataReader with the updates for this uploaded table data.

Remarks

The insert, update and delete operations are all accessible via the standard ADO.NET IDataReader. The tables metadata can be accessed via the GetSchemaTable call or the insert and delete data readers. The delete data reader only includes the primary key columns of the table.

GetDeletes method

Gets a DataReader with the deletes for this uploaded table data.

Visual Basic syntax

```
Public Function GetDeletes() As IDataReader
```

C# syntax

```
public IDataReader GetDeletes()
```

Returns

A DataReader with primary key columns for deleted rows.

Remarks

Each delete is represented by the primary key values needed to uniquely represent a row in this instances table.

Note
The index and order of the columns match the array for property DataTable.PrimaryKey for the schema of this table.

Example

Assuming that your remote client contains a table called sparse_pk, the following example uses the GetDeletes method to obtain a data reader of deleted rows. In this case, the delete DataReader includes two primary key columns. Note the index of each primary key column.

```
CREATE TABLE sparse_pk (
    pcol1  INT NOT NULL,
    col2 VARCHAR(200),
    pcol3 INT NOT NULL,
    PRIMARY KEY (pcol1, pcol3)
);

using iAnywhere.MobiLink.Script;
using System;
using System.IO;
using System.Data;
using System.Text;
...

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {

    // Get an UploadedTableData for the sparse_pk table.
    UploadedTableData sparse_pk_table =
ut.GetUploadedTableByName("sparse_pk");

    // Get deletes uploaded by the MobiLink client.
    using( IDataReader data_reader = sparse_pk_table.GetDeletes() ) {
            while (data_reader.Read()) {
            StringBuilder row_str = new StringBuilder("( ");
            row_str.Append(data_reader.GetString(0)); // pcol1
            row_str.Append(", ");
```

```
                row_str.Append(data_reader.GetString(1)); // pcol3
                row_str.Append(" )");
                writer.WriteLine(row_str);
                }
        }
    }
```

GetInserts method

Gets a DataReader with the inserts for this uploaded table data.

Visual Basic syntax

```
Public Function GetInserts() As IDataReader
```

C# syntax

```
public IDataReader GetInserts()
```

Returns

A DataReader with inserts for this table data.

Remarks

Each insert is represented by one row in the result set.

Example

```
CREATE TABLE sparse_pk (
    pcol1  INT NOT NULL,
    col2 VARCHAR(200),
    pcol3 INT NOT NULL,
    PRIMARY KEY (pcol1, pcol3)
);

using iAnywhere.MobiLink.Script;
using System;
using System.IO;
using System.Data;
using System.Text;
...

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {

    // Get an UploadedTableData for the sparse_pk table.
    UploadedTableData sparse_pk_table =
ut.GetUploadedTableByName("sparse_pk");

    // Get inserts uploaded by the MobiLink client.
    using( IDataReader data_reader = sparse_pk_table.GetInserts() ) {

        while (data_reader.Read()) {
            StringBuilder row_str = new StringBuilder("( ");
            row_str.Append(data_reader.GetString(0)); // pcol1
            row_str.Append(", ");
            if (data_reader.IsDBNull(1)) {
                row_str.Append("&lt;NULL&gt;");
            }
```



```
                else {
                    row_str.Append(data_reader.GetString(1)); // col2
                }
                row_str.Append(", ");
                row_str.Append(data_reader.GetString(2)); // pcol3
                row_str.Append(" )");
                writer.WriteLine(row_str);
            }
        }
    }
```

GetName method

Gets the table name of this instance.

Visual Basic syntax

```
Public Function GetName() As String
```

C# syntax

```
public string GetName()
```

Returns

The table name of this instance.

Remarks

This is a utility function. The table name can also be accessed via the Schema for this instance.

GetSchemaTable method

Gets a DataTable that describes the metadata for this download table.

Visual Basic syntax

```
Public Function GetSchemaTable() As DataTable
```

C# syntax

```
public DataTable GetSchemaTable()
```

Returns

A DataTable that describes the column metadata.

Remarks

If you want the DataTable to contain column name information, you must specify the client option to send column names, which is the default behavior. For more information on sending column names, see "SendColumnNames (scn) extended option" [*MobiLink - Client Administration*] and "Send Column Names synchronization parameter" [*UltraLite - Database Management and Reference*].

GetUpdates method

Gets a DataReader with the updates for this uploaded table data.

Visual Basic syntax

```
Public Function GetUpdates() As UpdateDataReader
```

C# syntax

```
public UpdateDataReader GetUpdates()
```

Returns

A DataReader with updates for this table data

Remarks

Each row in the result set represent one update. The mode of the result set can be flipped between new and old column values.

Example

The following example illustrates how to use the GetUpdates method:

```
CREATE TABLE sparse_pk (
    pcol1  INT NOT NULL,
    col2 VARCHAR(200),
    pcol3 INT NOT NULL,
    PRIMARY KEY (pcol1, pcol3)
);

using iAnywhere.MobiLink.Script;
using System;
using System.IO;
using System.Data;
using System.Text;
...

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {

    // Get an UploadedTableData for the sparse_pk table.
    UploadedTableData sparse_pk_table =
ut.GetUploadedTableByName("sparse_pk");

    // Get updates uploaded by the MobiLink client.
    using( UpdateDataReader data_reader = sparse_pk_table.GetInserts() ) {

        while (data_reader.Read()) {
            data_reader.SetNewRowValues();
            StringBuilder row_str = new StringBuilder("New values ( ");
            row_str.Append(data_reader.GetString(0)); // pcol1
            row_str.Append(", ");
            if (data_reader.IsDBNull(1)) {
                row_str.Append("&lt;NULL&gt;");
            }
            else {
                row_str.Append(data_reader.GetString(1)); // col2
            }
            row_str.Append(", ");
```



```
                row_str.Append(data_reader.GetString(2)); // pcol3
                row_str.Append(" )");
                data_reader.SetOldRowValues();
                row_str.Append(" Old Values (");
                row_str.Append(data_reader.GetString(0)); // pcol1
                row_str.Append(", ");
                if (data_reader.IsDBNull(1)) {
                    row_str.Append("&lt;NULL&gt;");
                }
                else {
                    row_str.Append(data_reader.GetString(1)); // col2
                }
                row_str.Append(", ");
                row_str.Append(data_reader.GetString(2)); // pcol3
                row_str.Append(" )");
                writer.WriteLine(row_str);
            }
        }
    }
```

LogCallback delegate

Called when the MobiLink server prints a message.

Visual Basic syntax

```
Public Delegate Sub LogCallback(
    ByVal sc As ServerContext,
    ByVal message As LogMessage
)
```

C# syntax

```
public delegate void LogCallback(ServerContext sc, LogMessage message);
```

Remarks

Implementations of this delegate can be registered with the ServerContext events to be called when the MobiLink server prints a message.

See also

- “ErrorListener event” on page 651
- “InfoListener event” on page 651
- “WarningListener event” on page 652

ShutdownCallback delegate

Called when MobiLink server is shutting down.

Visual Basic syntax

```
Public Delegate Sub ShutdownCallback(ByVal sc As ServerContext)
```

C# syntax

```
public delegate void ShutdownCallback(ServerContext sc);
```

Remarks

Implementations of this delegate can be registered with the ShutdownListener event to be called when the MobiLink server shuts down.

See also

- "ShutdownListener event" on page 652

SQLType enumeration

Enumerates all possible ODBC data types.

Visual Basic syntax

```
Public Enum SQLType
```

C# syntax

```
public enum SQLType
```

Members

Member name	Description
SQL_TYPE_NULL	Null data type.
SQL_UN-KNOWN_TYPE	Unknown data type.
SQL_CHAR	UTF-8 character array of a set size. Has .NET type String.
SQL_NUMERIC	Numeric value of set size and precision. Has .NET type Decimal.
SQL_DECIMAL	Decimal number of set size and precision. Has .NET type Decimal.
SQL_INTEGER	32-bit integer. Has .NET type Int32.

Member name	Description
SQL_SMALLINT	16-bit integer. Has .NET type Int16
SQL_FLOAT	Floating point number with ODBC driver defined precision. Has .NET type Double.
SQL_REAL	Single precision floating point number. Has .NET type Single.
SQL_DOUBLE	Double precision floating point number. Has .NET type Double.
SQL_DATE	A date. Has .NET type DateTime.
SQL_DATETIME	A date and time. Has .NET type DateTime.
SQL_TIME	A time. Has .NET type DateTime
SQL_INTERVAL	An interval of time. Has .NET type TimeSpan
SQL_TIMESTAMP	A time stamp. Has .NET type DateTime.
SQL_VARCHAR	A null terminated UTF-8 string with a user set maximum length. Has .NET type String.
SQL_TYPE_DATE	A date. Has .NET type DateTime.
SQL_TYPE_TIME	A time. Has .NET type DateTime.

Member name	Description
SQL_TYPE_TIME-STAMP	A timestamp. Has .NET type DateTime
SQL_DEFAULT	A default type. Has no type.
SQL_ARD_TYPE	An ARD object. Has no type.
SQL_BIT	A single bit. Has .NET type Boolean.
SQL_TINYINT	8-bit integer. Has .NET type SByte.
SQL_BIGINT	64-bit integer. Has .NET type Int64.
SQL_LONGVARBINA-RY	Variable length binary data with a driver dependent maximum length. Has .NET type byte[].
SQL_VARBINARY	Variable length binary data with a user specified maximum length. Has .NET type byte[].
SQL_BINARY	Fixed length binary data. Has .NET type byte[].
SQL_LONGVARCHAR	A null terminated UTF-8 string with a driver dependent maximum length. Has .NET type String.
SQL_GUID	A GUID. Has .NET type Guid.
SQL_WCHAR	Unicode character array of fixed size. Has .NET type String.

Member name	Description
SQL_WVARCHAR	Null terminated Unicode string of user defined maximum length; Has .NET type String.
SQL_WLONGVARCH-AR	Null terminated Unicode string of driver dependent maximum length; Has .NET type String.
SQL_SS_TIMESTAM-POFFSET	Timestamp with time zone offset; Has .NET type iAnywhere.Mobi-Link.Script.DateTimeWithTimeZone This can be used only against Micro-soft SQL Server and Oracle databases.

Remarks

Each SQLType corresponds to a .NET type.

Direct row handling

Note

Direct row handling is an advanced MobiLink feature. To use it, you must have a thorough understanding of how to create a MobiLink application and how to use the MobiLink APIs. See:

- "MobiLink - Getting Started"
- "MobiLink - Server Administration" on page 1
- "MobiLink - Client Administration"

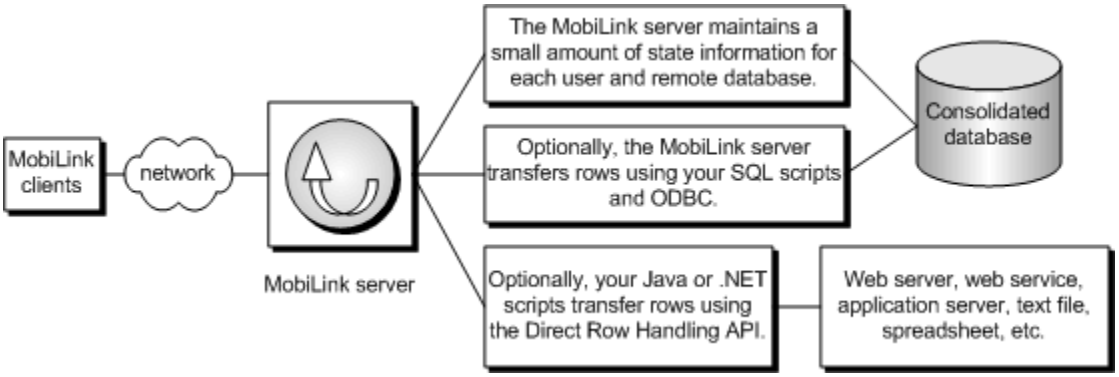
MobiLink supports two ways to handle rows: SQL and direct. You can use them separately or together.

- **SQL row handling** allows you to synchronize remote data to a supported consolidated database. SQL-based events provide a robust interface for conflict resolution and other synchronization tasks. You can use SQL directly or you can return SQL using the MobiLink server APIs for Java and .NET.
- **Direct row handling** allows you to synchronize remote data with any central data source. Direct row handling allows you to access raw synchronized data using special MobiLink events and the MobiLink server APIs for Java and .NET.

  The data sources you can synchronize can be virtually anything, including an application, web server, web service, application server, text file, spreadsheet, non-relational database, or an RDBMS that cannot be used as a consolidated database. You still need a consolidated database to store your MobiLink system tables, and many implementations of direct row handling synchronizes to both the consolidated database and another data source.

  To use direct row handling, you need familiarity with how to create a MobiLink consolidated database, add synchronization scripts, and create MobiLink remote users.

The following diagram shows the basic MobiLink architecture:



The components of direct row handling

To implement direct row handling, you can use two synchronization events along with several interfaces and methods in the MobiLink server APIs for Java and .NET.

Direct synchronization events

Direct row handling allows you to directly access the upload stream and download stream. You do this by writing Java or .NET methods for the handle_UploadData and handle_DownloadData synchronization events.

- **handle_UploadData** accepts a single UploadData parameter that encapsulates operations uploaded by a MobiLink client for a single upload transaction. See:
  - “Handling direct uploads” on page 675
  - “handle_UploadData connection event” on page 394
- **handle_DownloadData** allows you to set download operations using the DownloadData interface. See:
  - “Handling direct downloads” on page 681
  - “handle_DownloadData connection event” on page 383

Components of the MobiLink server API for direct row handling

For the Java API:

- DBConnectionContext “getDownloadData method” on page 538
- “DownloadData interface” on page 542
- “DownloadTableData interface” on page 544
- “UpdateResultSet” on page 578
- “UploadData interface” on page 580
- “UploadedTableData interface” on page 582

For the .NET API:

- DBConnectionContext "GetDownloadData method" on page 616
- "DownloadData interface" on page 635
- "DownloadTableData interface" on page 637
- "UpdateDataReader interface" on page 658
- "UploadedTableData interface" on page 662
- "UploadData interface" on page 660

Set up direct row handling

To use direct row handling, you need familiarity with how to create a MobiLink consolidated database, add synchronization scripts, and create MobiLink remote users.

To synchronize with a data source other than a consolidated database, complete the following steps.

Set up direct row handling

1. Set up a consolidated database, if you do not already have one.

   Whether you are synchronizing to a consolidated database, you need to have a consolidated database to hold MobiLink system tables.

   See "MobiLink consolidated databases" on page 1.

2. If you want to handle uploads, write a public method using the UploadData interface and register it for the handle_UploadData connection event.

   See "Handling direct uploads" on page 675.

3. If you want to handle downloads, write a public method using the DownloadData interface and register it for the handle_DownloadData connection event (or another event).

   See "Handling direct downloads" on page 681.

4. If you want to use the row handling API to refer to columns by name (rather than by index), specify in your client that column names should be sent with the upload. See:
   - SQL Anywhere clients: "SendColumnNames (scn) extended option" [*MobiLink - Client Administration*]
   - UltraLite: "Send Column Names synchronization parameter" [*UltraLite - Database Management and Reference*]

Other resources for getting started

- "Tutorial: Introduction to direct row handling" [*MobiLink - Getting Started*]
- http://www.sybase.com/detail?id=1058600#319
- "Setting up Java synchronization logic" on page 523
- "Setting up .NET synchronization logic" on page 588

You can post questions on the MobiLink newsgroup: sybase.public.sqlanywhere.mobilink.

Development tips for direct row handling

Unique primary keys

For MobiLink synchronization, including direct row handling, your data source must have unique primary keys that are not updated. In a non-relational data source such as a spreadsheet or text file, this means that one column must contain unique, unchanging values that identify the row.

See "Maintaining unique primary keys" on page 96.

Column Names

When using direct row handling, the column names of tables are only available if the MobiLink client is configured to send column names. Alternatively, you can use column indexes to access row information.

To use column names, see:

- SQL Anywhere remotes: "SendColumnNames (scn) extended option" [*MobiLink - Client Administration*]
- UltraLite remotes: "Send Column Names synchronization parameter" [*UltraLite - Database Management and Reference*]

Use the last download time for downloads

If possible, set up your direct row handling application like a timestamp-based SQL application; maintain a last_modified column and download data based on it. This method avoids unforeseen problems that could occur if you use a different download methodology.

See "Timestamp-based downloads" on page 86.

Transaction management for uploads

You cannot commit transactions with the MobiLink consolidated database. However, you can commit transactions with your direct row handling data source. When setting up transaction management, keep the following tips in mind:

- **Commit the upload before MobiLink commits** When applying an upload, MobiLink commits the changes at the end of the end_upload event. You should make sure that all upload changes that you want to keep are committed before the end of your end_upload script. Otherwise, if there is an error or failure you may get into a state in which your application thinks that the upload is applied but MobiLink has not applied the data, which could result in lost data.

- **Handle redundant uploads** When an error or failure occurs after your application commits an uploaded row and before the MobiLink server commits it, the MobiLink server and your data source may get in an inconsistent state. You can solve this problem by allowing redundant uploads and having logic in place to make sure the redundant upload is applied properly. In particular, when your application sends the upload a second time, it should not be applied again.

Handle errors

To handle errors, ensure you employ appropriate transaction management, as described above. In addition, your Java or .NET code that handles rows must send any exception that occurs to the MobiLink server. If an error occurs before the MobiLink server or your application has committed changes, MobiLink rollbacks the transaction and maintains a consistent state with your application.

Class instance

For direct row handling, MobiLink creates one class instance per database connection. The class instance is not destroyed at the end of a synchronization: it is destroyed when the database connection is closed. Class level variables retain values from previous synchronizations.

Handling direct uploads

To handle direct uploads, complete the following steps:

Handle direct uploads

1. Register a Java or .NET method for the "handle_UploadData connection event" on page 394.
2. Write a method for the handle_UploadData synchronization event. This event accepts one UploadData parameter. See:
   - Java server API: "UploadData interface" on page 580
   - .NET server API: "UploadData interface" on page 660

The handle_UploadData event is usually called once per synchronization. However, for SQL Anywhere clients that use transaction-level uploads, there can be more than one upload per synchronization, in which case handle_UploadData is called once per transaction.

For more information about dbmlsync transaction-level uploads, see "-tu dbmlsync option" [*MobiLink - Client Administration*].

For general information about writing Java or .NET synchronization scripts, see:

- "Writing synchronization scripts in Java" on page 523
- "Writing synchronization scripts in .NET" on page 588

For information about registering connection-level events, see:

- "ml_add_java_connection_script system procedure" on page 698
- "ml_add_dnet_connection_script system procedure" on page 695

Classes for direct uploads

The MobiLink server APIs for Java and .NET provide the following interfaces for handling direct uploads:

- **UploadData** Encapsulates a single upload transaction. An upload transaction contains a set of tables containing row operations. See:
  - Java API: "UploadData interface" on page 580
  - .NET API: "UploadData interface" on page 660
- **UploadedTableData** Encapsulates a table's insert, update, and delete operations uploaded by a MobiLink client. For Java, UploadedTableData methods return an instance of an UpdateResultSet. For .NET, UploadedTableData methods return an instance of an UpdateDataReader interface. You traverse the result set IDataReader to process the uploaded row operations. See:
  - Java API: "UploadedTableData interface" on page 582
  - .NET API: "UploadedTableData interface" on page 662
- **UpdateResultSet** For Java, this class represents an update result set returned by the UploadedTableData getUpdates method. It extends java.sql.ResultSet to include special methods for retrieving the new and old versions of an updated row.

  See "UpdateResultSet" on page 578.

  For .NET, the UpdateDataReader interface represents a set of rows returned by the UploadedTableData GetUpdates method. It extends IDataReader to include special methods for retrieving the new and old versions of an updated row.

  See "UpdateDataReader interface" on page 658.

Example

See "handle_UploadData connection event" on page 394.

Handling direct upload conflicts

When a MobiLink client sends an updated row to the MobiLink server, it includes not only the updated values (the post-image or new row), but also a copy of the old row values (the pre-image or old row) obtained in the last synchronization with the MobiLink server. When the pre-image row does not match the current values in your central data source, a conflict is detected.

SQL-based conflict resolution

For SQL-based uploads, the MobiLink consolidated database is your central data source and MobiLink provides special events for conflict detection and resolution.

See "Handling conflicts" on page 102.

Conflict resolution with direct row handling

For direct uploads, you can access new and old rows programmatically for conflict detection and resolution.

UpdateResultSet (returned by the UploadedTableData.getUpdates method) extends standard Java or .NET result sets to include special methods for handling conflicts. setNewRowValues sets UpdateResultSet to

return new updated values from a remote client (the default mode). setOldRowValues sets UpdateResultSet to return old row values.

Detecting conflicts with direct row handling

By using the UpdateResultSet method .setOldRowValues, you get the values of a row on the remote before it was changed. You compare the row values that are returned to the existing row values in your data source. If the rows you compare are not equal, then a conflict exists.

Resolving conflicts with direct row handling

Once you have detected a conflict during an upload, you can use custom business logic to resolve the conflict. The resolution is handled by your Java or .NET code.

Example

Suppose you track inventory in an XML document and want to use it as your central data source. User1 uses one of your remote databases called Remote1. User2 uses another remote database called Remote2.

Your XML document, User1, and User2 all start with an inventory of ten items. User1 sells three items and updates the Remote1 inventory value to seven items. User2 sells four items and updates the Remote2 inventory to six items. When Remote1 synchronizes, the central database is updated to seven items. When Remote2 synchronizes, a conflict is detected because the value of the inventory is no longer ten items. To resolve this conflict programmatically, you need three row values:

- The current value in the central data source.
- The new row value that Remote2 uploaded.
- The old row value that Remote2 obtained during the last synchronization.

In this case, the business logic would use the following formula to calculate the new inventory value and resolve the conflict:

```
current data source - (old remote - new remote)
-> 7 - (10-6) = 3
```

The following procedures for Java and .NET demonstrate how you can resolve this conflict for direct uploads, using the following table as an example:

```
CREATE TABLE remoteOrders
(
    pk integer primary key not null,
    inventory integer not null
);
```

Handle direct conflicts (Java)

1. Register a Java or .NET method for the handle_UploadData connection event.

   See “handle_UploadData connection event” on page 394.

   For example, the following stored procedure call registers a Java method called HandleUpload for the handle_UploadData connection event when synchronizing the script version ver1. You run this stored procedure against your MobiLink consolidated database.

```
call ml_add_java_connection_script( 'ver1',
   'handle_UploadData',
   'OrderProcessor.HandleUpload' )
```

For more information about registering methods for synchronization events, see:

- "Adding and deleting scripts" on page 285
- "ml_add_java_connection_script system procedure" on page 698

2. Obtain an UpdateResultSet for a table in the upload.

   The OrderProcessor.HandleUpload method obtains an UpdateResultSet for the remoteOrders table:

```
// method for handle_UploadData event
 public void HandleUpload( UploadData u_data )
 {

    // Get UploadedTableData for the remoteOrders table.
      UploadedTableData u_table =
u_data.getUploadedTableByName("remoteOrders");

    // Get an UpdateResultSet for the remoteOrders table.
    UpdateResultSet update_rs = u_table.getUpdates();

    // (Continued...)
```

3. For each update, get the current values in your central data source.

   In this example, the UpdateResultSet getInt method returns an integer value for the primary key column (the first column). You can implement and then use the getMyCentralData method to get data from your central data source.

```
while( update_rs.next() )
{
   // Get central data source values.

   // Get the primary key value.
   int pk_value = update_rs.getInt(1);

   // Get central data source values.
   int central_value = getMyCentralData(pk_value);

   // (Continued...)
```

4. For each update, get the old and new values uploaded by the MobiLink client.

   The example uses the UpdateResultSet setOldRowValues and UpdateResultSet setNewRowValues for old and new values, respectively.

```
   // Set mode for old row values.
   update_rs.setOldRowValues();

   // Get the _old_ stored value on the remote.
   int old_value = update_rs.getInt(2);

   // Set mode for new row values.
   update_rs.setNewRowValues();

   // Get the _new_ updated value on the remote.
   int new_value = update_rs.getInt(2);
```



```
    // (Continued...)
```

5. For each update, check for conflicts.

   A conflict occurs when the old row value does not match the current value in the central data source. To resolve the conflict, a resolved value is calculated using business logic. If no conflict occurs, the central data source is updated with the new remote value. You can implement and then use the setMyCentralData method to perform the update.

```
    // Check if there is a conflict.

    if(old_value == central_value)
    {
      // No conflict.
      setMyCentralData(pk_value, new_value);

    }
    else
    {
      // Handle the conflict.
      int inventory = old_value - new_value;
      int resolved_value = central_value - inventory;

      setMyCentralData(pk_value, resolved_value);

    }
  }
```

Handle direct conflicts (.NET)

1. Register a method for the handle_UploadData connection event.

   For example, the following stored procedure call registers a .NET method called HandleUpload for the handle_UploadData connection event when synchronizing the script version ver1. You run this stored procedure against your MobiLink consolidated database.

```
call ml_add_dnet_connection_script( 'ver1',
   'handle_UploadData',
   'MyScripts.OrderProcessor.HandleUpload' )
```

   For more information about registering methods for synchronization events, see:

   - “Adding and deleting scripts” on page 285
   - “ml_add_dnet_connection_script system procedure” on page 695

2. Obtain an UpdateDataReader for a table in the upload.

   The MyScripts.OrderProcessor.HandleUpload method obtains an UpdateResultSet for the remoteOrders table:

```
// method for handle_UploadData event
 public void HandleUpload( UploadData u_data )
 {

    // Get UploadedTableData for the remoteOrders table.
      UploadedTableData u_table =
u_data.GetUploadedTableByName("remoteOrders");
```

```
        // Get an UpdateDataReader for the remoteOrders table.
        UpdateDataReader update_dr = u_table.GetUpdates();

        // (Continued...)
```

3. For each update, get the current values in your central data source.

   In this example, the UpdateDataReader GetInt32 method returns an integer value for the primary key column (the first column). You can implement and then use the getMyCentralData method to get data from your central data source.

```
while( update_dr.Read() )
{
   // Get central data source values.

   // Get the primary key value.
   int pk_value = update_dr.GetInt32(0);

   // Get central data source values.
   int central_value = getMyCentralData(pk_value);

   // (Continued...)
```

4. For each update, get the old and new values uploaded by the MobiLink client.

   The example uses the UpdateResultSet setOldRowValues and UpdateResultSet setNewRowValues for old and new values, respectively.

```
   // Set mode for old row values.
   update_dr.SetOldRowValues();

   // Get an _old_ value.
   int old_value = update_dr.GetInt32(1);

   // Set mode for new row values.
   update_dr.SetNewRowValues();

   // Get the _new_ updated value.
   int new_value = update_dr.GetInt32(1);

   // (Continued...)
```

5. For each update, check for conflicts.

   A conflict occurs when the old row value does not match the current value in the central data source. To resolve the conflict, a resolved value is calculated using business logic. If no conflict occurs, the central data source is updated with the new remote value. You can implement and then use the setMyCentralData method to perform the update.

```
   // Check if there is a conflict.

   if(old_value == central_value)
   {
     // No conflict.
     setMyCentralData(pk_value, new_value);

   }
   else
```



```
    {
      // Handle the conflict.
      int inventory = old_value - new_value;
      int resolved_value = central_value - inventory;

      setMyCentralData(pk_value, resolved_value);

    }
  }
```

Handling direct downloads

To handle direct downloads, complete the following steps:

Handle direct downloads

1. Register a Java or .NET method for the "handle_DownloadData connection event" on page 383.

2. Write a method for the handle_DownloadData synchronization event. In this event you use an instance of DBConnectionContext to get a DowloadData instance for the current synchronization. See:

   - Java: "DBConnectionContext interface" on page 536
   - Java: "DownloadData interface" on page 542
   - .NET: "DBConnectionContext interface" on page 614
   - .NET: "DownloadData interface" on page 635

You can create the entire direct download in the handle_DownloadData synchronization event. Alternatively, you can use other synchronization events to set direct download operations. However, you must create a handle_DownloadData script, even if its method does nothing. If you process the direct download in an event other than handle_DownloadData, the event cannot be before begin_synchronization and cannot be after end_download.

For information about the order of events, see "MobiLink complete event model" on page 301.

Classes for direct downloads

The MobiLink server APIs for Java and .NET provide the following classes for creating direct downloads:

- **DownloadData** Encapsulates download tables containing operations to send down to a remote client during synchronization. See:

  - Java: "DownloadData interface" on page 542
  - .NET: "DownloadData interface" on page 635

- **DownloadTableData** Encapsulates upsert (update and insert) and delete operations to download to a MobiLink client.

  For Java, DownloadTableData methods return an instance of a JDBC PreparedStatement. In Java, you add a row to the download by setting the prepared statement's column values and then executing the prepared statement.

For .NET, DownloadTableData methods return an instance of a .NET IDbCommand. In .NET, you add a row to the download by setting the command's column values and then executing the command.

See:

- Java: “DownloadTableData interface” on page 544
- .NET: “DownloadTableData interface” on page 637

Example

See “handle_DownloadData connection event” on page 383.

MobiLink reference

This section contains MobiLink reference material.

MobiLink Replay C++ callback methods

This section describes the public structure of the *mlreplaycallbacks.cpp* file, and contains a complete list of callback methods that you can develop to customize the data uploaded to the MobiLink server during a replay session using the mlgenreplay utility.

CreateAndInitMLReplayUploadTransaction method

Called once per MLReplay instance; use to create and initialize an upload transaction.

Syntax

```
public bool _callback CreateAndInitMLReplayUploadTransaction(
    IMLReplayUploadTransaction ** uploadTrans,
    MLReplayInfo * mlrInfo
)
```

Parameters

- **uploadTrans** An implementation of IMLReplayUploadTransaction that MLReplay uses to populate the replay session with custom data.
- **mlrInfo** Information from MLReplay.

Returns

True on success; returns false on error, which cancels the replay session.

DestroyMLReplayUploadTransaction method

Called once per MLReplay instance; use to de-construct an upload transaction.

Syntax

```
public void _callback DestroyMLReplayUploadTransaction(
    IMLReplayUploadTransaction * uploadTrans
)
```

Parameters

- **uploadTrans** An implementation of IMLReplayUploadTransaction that MLReplay used to populate the replay session with custom data.

FiniIdentifySimulatedClient method

Called once per simulated client; use to clean up memory used by the call to IdentifySimulatedClient for the given simulated client.

Syntax

```
public void _callback FiniIdentifySimulatedClient(
    asa_uint32 simulatedClientNum,
    asa_uint32 numSimulatedClients,
    char * remoteID,
    char * username,
    char * password,
    char ** authenticationParameters,
    asa_uint16 numAuthenticationParameters,
    char * ldt,
    const MLReplayNamesAndValues * namesAndValues
)
```

Parameters

- **simulatedClientNum** The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same MLReplay instance.
- **numSimulatedClients** The total number of simulated clients in this MLReplay instance.
- **remoteID** The remote ID given by the call to IdentifySimulatedClient for the given simulated client.
- **username** The username given by the call to IdentifySimulatedClient for the given simulated client.
- **password** The password given by the call to IdentifySimulatedClient for the given simulated client.
- **authenticationParameters** The authentication parameters given by the call to IdentifySimulatedClient for the given simulated client.
- **numAuthenticationParameters** The number of authentication parameters given by the call to IdentifySimulatedClient for the given simulated client.
- **ldt** The LDT given by the call to IdentifySimulatedClient for the given simulated client.
- **namesAndValues** The set of defined names and values used to customize replay behavior.

FreeAllUploadRows method

Called once per upload transaction, per synchronization, and per simulated client; use to free all upload rows in the given IMLReplayUploadTransaction.

Syntax

```
public void _callback FreeAllUploadRows(
    IMLReplayUploadTransaction * uploadTrans
)
```


Parameters

- **uploadTrans** An implementation of IMLReplayUploadTransaction that MLReplay uses to populate the replay session with custom data.

GetDownloadApplyTime method

Called once per download, per simulated client; use to simulate slow devices.

Syntax

```
public asa_uint32 _callback GetDownloadApplyTime(
    asa_uint32 repetitionNum,
    asa_uint32 numRepetitions,
    asa_uint32 simulatedClientNum,
    asa_uint32 numSimulatedClients,
    asa_uint32 recordedSyncNum,
    asa_uint32 numRecordedSyncs,
    asa_uint32 recordedDownloadApplyTime,
    const MLReplayNamesAndValues * namesAndValues
)
```

Parameters

- **repetitionNum** The current repetition number, which is always 1.
- **numRepetitions** The number of times to repeat the session, which is always 1.
- **simulatedClientNum** The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same MLReplay instance.
- **numSimulatedClients** The total number of simulated clients in this MLReplay instance.
- **recordedSyncNum** The synchronization number (ordinal 1) within the recorded protocol.
- **numRecordedSyncs** The total number of synchronizations in the recorded protocol.
- **recordedDownloadApplyTime** The recorded download apply time (in milliseconds).
- **namesAndValues** The set of defined names and values used to customize replay behavior.

Returns

The number of milliseconds it should take to apply the download.

Remarks

MLReplay does a good job of estimating the download apply time for synchronizations that do not occur in a persistent connection. For synchronizations that occur in a persistent connection, MLReplay cannot accurately estimate the download apply time unless download acknowledgements are used.

GetNumRows method

Called once per table, per upload transaction, per synchronization, per simulated client; use to return the number of rows in the given IMLReplayUploadTable.

Syntax

```
public asa_uint32 _callback GetNumRows(
    const IMLReplayUploadTable * uploadTable
)
```

Parameters

- **uploadTable** An implementation of IMLReplayUploadTransaction that MLReplay uses to populate the replay session with custom data.

Returns

The number of rows in the given IMLReplayUploadTable.

GetNumUploadTables method

Called once per upload transaction, per synchronization, per simulated client; use to return the number of tables in the given IMLReplayUploadTransaction.

Syntax

```
public asa_uint32 _callback GetNumUploadTables(
    const IMLReplayUploadTransaction * uploadTrans
)
```

Parameters

- **uploadTrans** An implementation of IMLReplayUploadTransaction that MLReplay uses to populate the replay session with custom data.

Returns

The number of tables in the given IMLReplayUploadTransaction.

GetRow method

Called once per row, per table, per upload transaction, per synchronization, per simulated client; use to return the requested row from the given IMLReplayUploadTable.

Syntax

```
public const MLReplayRow *_callback GetRow(
    const IMLReplayUploadTable * uploadTable,
    asa_uint32 rowNum
)
```


Parameters

- **uploadTable** An implementation of IMLReplayUploadTransaction that MLReplay uses to populate the replay session with custom data.
- **rowNum** The requested row number.

Returns

The requested row from the given IMLReplayUploadTable.

GetUploadTable method

Called once per table, per upload transaction, per synchronization, per simulated client; use to return the table with the given index.

Syntax

```
public const IMLReplayUploadTable *_callback GetUploadTable(
    const IMLReplayUploadTransaction * uploadTrans,
    asa_uint32 tableIndex
)
```

Parameters

- **uploadTrans** An implementation of IMLReplayUploadTransaction that MLReplay uses to populate the replay session with custom data.
- **tableIndex** The index of the requested table.

Returns

The table with the given index.

GetUploadTransaction method

Called once per upload transaction, per synchronization, per simulated client; use to return true on success or false on error, which cancels the replay session.

Syntax

```
public bool _callback GetUploadTransaction(
    asa_uint32 repetitionNum,
    asa_uint32 numRepetitions,
    asa_uint32 simulatedClientNum,
    asa_uint32 numSimulatedClients,
    asa_uint32 syncNum,
    asa_uint32 numRecordedSyncs,
    asa_uint32 transNum,
    asa_uint32 numUploadedTrans,
    IMLReplayUploadTransaction * uploadTrans,
```

```
    const MLReplayNamesAndValues * namesAndValues
)
```

Parameters

- **repetitionNum** The current repetition number, which is always 1.
- **numRepetitions** The number of times to repeat the session, which is always 1.
- **simulatedClientNum** The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same MLReplay instance.
- **numSimulatedClients** The total number of simulated clients in this MLReplay instance.
- **syncNum** The synchronization number (ordinal 1) within the recorded protocol.
- **numRecordedSyncs** The total number of synchronizations in the recorded protocol.
- **transNum** The transaction number (ordinal 1) within the given synchronization.
- **numUploadedTrans** The total number of upload transactions in the given synchronization.
- **uploadTrans** An output parameter that must be set with the upload operations for the current transaction.
- **namesAndValues** The set of defined names and values used to customize replay behavior.

Returns

True on success, false on error, which cancels the replay session.

GlobalFini method

Called once per MLReplay instance; use to clean up any global variables used by the other callbacks.

Syntax

```
public void _callback GlobalFini(
    const MLReplayNamesAndValues * namesAndValues
)
```

Parameters

- **namesAndValues** The set of defined names and values used to customize replay behavior.

GlobalInit method

Called once per MLReplay instance; use to initialize any global variables used by the other callbacks.

Syntax

```
public bool _callback GlobalInit(
    const MLReplayNamesAndValues * namesAndValues
)
```

Parameters

- **namesAndValues** The set of defined names and values used to customize replay behavior.

Returns

True on success; false on error, which cancels the replay session.

IdentifySimulatedClient method

Called once per simulated client; use to specify the simulated client information.

Syntax

```
public bool _callback IdentifySimulatedClient(
    asa_uint32 simulatedClientNum,
    asa_uint32 numSimulatedClients,
    char ** remoteID,
    char ** username,
    char ** password,
    char *** authenticationParameters,
    asa_uint16 * numAuthenticationParameters,
    char ** ldt,
    const MLReplayNamesAndValues * namesAndValues
)
```

Parameters

- **simulatedClientNum** The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same MLReplay instance.
- **numSimulatedClients** The total number of simulated clients in this MLReplay instance.
- **remoteID** An output parameter that must be set to the remote ID of this simulated client, which must be a unique value across all mlreplay instances.
- **username** An output parameter that must be set to the MobiLink username for this simulated client.
- **password** An output parameter that must be set to the password for the MobiLink user.
- **authenticationParameters** An output parameter that must be set to an array of authentication parameters for this simulated client.
- **numAuthenticationParameters** An output parameter set to the number of authentication parameters returned in authenticationParameters.

- **ldt** An output parameter set to the LDT for the user. The format of ldt must be yyyy-MM-dd hh:mm:ss.SSS.
- **namesAndValues** The set of defined names and values used to customize replay behavior.

Remarks

If the username, password, authenticationParameters, or ldt are null, then MLReplay uses the corresponding values in the recorded protocol. If remoteID is null, MLReplay replaces the remote ID with a GUID for the simulated client.

Returns

True on success, false on error, which cancels the replay session.

ReportEndOfSimulatedClient method

Called once per simulated client; use to perform any actions required when a simulated client is finished replaying.

Syntax

```
public bool _callback ReportEndOfSimulatedClient(
    asa_uint32 repetitionNum,
    asa_uint32 numRepetitions,
    asa_uint32 simulatedClientNum,
    asa_uint32 numSimulatedClients,
    bool success,
    const MLReplayNamesAndValues * namesAndValues
)
```

Parameters

- **repetitionNum** The current repetition number, which is always 1.
- **numRepetitions** The number of times to repeat the session, which is always 1.
- **simulatedClientNum** The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same MLReplay instance.
- **numSimulatedClients** The total number of simulated clients in this MLReplay instance.
- **success** True when the simulated client completed successfully; otherwise, false.
- **namesAndValues** The set of defined names and values used to customize replay behavior.

Returns

True on success; false on error.

Remarks

This function can be called concurrently by multiple simulated clients.

WaitForSimulatedClientStart method

Called once per simulated client; use to determine if the application should wait for a simulated client.

Syntax

```
public bool _callback WaitForSimulatedClientStart(
    asa_uint32 repetitionNum,
    asa_uint32 numRepetitions,
    asa_uint32 simulatedClientNum,
    asa_uint32 numSimulatedClients,
    const MLReplayNamesAndValues * namesAndValues
)
```

Parameters

- **repetitionNum** The current repetition number, which is always 1.
- **numRepetitions** The number of times to repeat the session, which is always 1.
- **simulatedClientNum** The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same MLReplay instance.
- **numSimulatedClients** The total number of simulated clients in this MLReplay instance.
- **namesAndValues** The set of defined names and values used to customize replay behavior.

Remarks

The replay session is cancelled when there is a return of false.

Example

Simulated client x starts as soon as WaitForSimulatedClientStart returns true for simulated clients 1,..., x. Therefore, WaitForSimulatedClientStart is not called for simulated client x until WaitForSimulatedClientStart returns for simulated clients 1,..., i-x.

MobiLink server system procedures

MobiLink provides the following stored procedures to help you create your applications.

System procedures to add or delete scripts

You must add synchronization scripts to system tables in the consolidated database before you can use them. The following system procedures add or delete synchronization scripts in the consolidated database:

- “ml_add_connection_script system procedure” on page 694
- “ml_add_table_script system procedure” on page 708
- “ml_add_dnet_connection_script system procedure” on page 695
- “ml_add_dnet_table_script system procedure” on page 697
- “ml_add_java_connection_script system procedure” on page 698
- “ml_add_java_table_script system procedure” on page 699

When you use the MobiLink server API for Java or .NET, you use these stored procedures to register a method as the script for an event, so that the method is run when the event occurs. You can also use them to unregister your methods.

When you add a script using a system procedure, the script is a string. Any strings within the script need to be escaped. For SQL Anywhere, each quotation mark (') needs to be doubled so as not to terminate the string.

You cannot use system procedures to add scripts longer than 255 bytes to Adaptive Server Enterprise 11.5 or earlier. Instead, use Sybase Central or direct insertion to define longer scripts.

IBM DB2 LUW before version 6 only supports column names and other identifiers of 18 characters or less, and so the names are truncated. For example, ml_add_connection_script is shortened to ml_add_connection_.

System procedures for managing remote tasks

The following stored procedures can be used to manage remote tasks:

Other system procedures

ml_add_column system procedure

Registers information about columns on remote databases for use by named column parameters.

Syntax

ml_add_column (
'*version*',
'*table*',
'*column*',
'*type*'
)

Parameters

Syntax	Description
version	VARCHAR(128). The version name.
table	VARCHAR(128). The table name.
column	VARCHAR(128). The column name.
type	VARCHAR(128). Reserved for future use. Set to null.

Remarks

This procedure populates the ml_column MobiLink system table with information about the columns on the remote database. The information is used by named row parameters.

Caution
ml_add_column calls must be executed in the same order that the columns exist in the remote database table. Failing to do so may result in incorrect data.

You need to run this system procedure if your synchronization clients **do not** send up column names. By default, version 12 synchronizations **do** send up column names, so ml_add_column is not required in most deployments. The ml_add_column names are overridden by names from the client, if they are supplied. Set the SendColumnNames (scn) extended option for dbmlsync to OFF if you do not want columns names to be sent from the client. See "SendColumnNames (scn) extended option" [*MobiLink - Client Administration*]

To delete all entries for the table name in the given script version, set the column name to null.

See also

- "Script parameters" on page 268

Examples

The following stored procedure call populates the ml_column MobiLink system table for col1 in MyTable for the script version Version1. This call allows you to use the named row parameters r.col1 and o.col1 in table scripts for MyTable1 in the Version1 script version when the synchronization client is not sending up column names (as clients prior to version 12 do by default).

```
CALL ml_add_column( 'Version1', 'MyTable1', 'col1', NULL )
```

The following stored procedure call deletes all entries in the ml_column MobiLink system table for MyTable1 in script version Version1:

```
CALL ml_add_column( 'Version1', 'MyTable1', NULL, NULL )
```

ml_add_connection_script system procedure

Use this system procedure to add or delete SQL connection scripts in the consolidated database.

Syntax

ml_add_connection_script (
'*version*',
'*event*',
'*script*'
)

Parameters

Syntax	Description
version	VARCHAR(128). The version name.
event	VARCHAR(128). The event name.
script	TEXT. The script contents. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For Oracle, this parameter is CLOB.

Remarks

To delete a connection script, set the script contents parameter to null.

When you add a script, the script is inserted into the ml_script table and the appropriate references are defined to associate the script with the event and script version that you specify. If the version name is new, it is automatically inserted into the ml_version table.

See also

- “System procedures to add or delete scripts” on page 691
- “Adding and deleting scripts” on page 285
- “ml_add_table_script system procedure” on page 708
- “ml_add_dnet_connection_script system procedure” on page 695
- “ml_add_dnet_table_script system procedure” on page 697
- “ml_add_java_connection_script system procedure” on page 698
- “ml_add_java_table_script system procedure” on page 699

Example

The following statement adds a connection script associated with the begin_synchronization event to the script version custdb in a SQL Anywhere consolidated database. The script itself is the single statement that sets the @EmployeeID variable.

```
call ml_add_connection_script( 'custdb',
 'begin_synchronization',
 'set @EmployeeID = {ml s.username}' )
```

ml_add_dnet_connection_script system procedure

Use this system procedure to register or unregister a .NET method as the script for a connection event.

Syntax

ml_add_dnet_connection_script (
'*version*',
'*event*',
'*script*'
)

Parameters

Syntax	Description
version	VARCHAR(128). The version name.
event	VARCHAR(128). The event name.
script	TEXT. The script contents. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For Oracle, this parameter is CLOB.

Remarks

To unregister a method, set the script contents parameter to null.

The script contents value is a public method in a class in a .NET assembly (for example, MyClass.MyMethod).

When you call ml_add_dnet_connection_script, the method is associated with the event and script version that you specify. If the version name is new, it is automatically inserted into the ml_version table.

See also

- “System procedures to add or delete scripts” on page 691
- “Adding and deleting scripts” on page 285
- “ml_add_dnet_table_script system procedure” on page 697
- “ml_add_connection_script system procedure” on page 694
- “ml_add_table_script system procedure” on page 708
- “ml_add_java_table_script system procedure” on page 699
- “Methods” on page 593
- “Writing synchronization scripts in .NET” on page 588

Example

The following example registers the beginDownloadConnection method of the ExampleClass class for the begin_download event.

```
call ml_add_dnet_connection_script( 'ver1',
'begin_download',
'ExamplePackage.ExampleClass.beginDownloadConnection' );
```


ml_add_dnet_table_script system procedure

Use this system procedure to register or unregister a .NET method as the script for a table event.

Syntax

ml_add_dnet_table_script (
'*version*',
'*table*',
'*event*',
'*script*'
)

Parameters

Syntax	Description
version	VARCHAR(128). The version name.
table	VARCHAR(128). The table name.
event	VARCHAR(128). The event name.
script	TEXT. The script contents. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For Oracle, this parameter is CLOB.

Remarks

To unregister a method, set the script contents parameter to null.

The script value is a public method in a class in a .NET assembly (for example, MyClass.MyMethod).

When you call ml_add_dnet_table_script, the method is associated with the table, event, and script version that you specify. If the version name is new, it is automatically inserted into the ml_version table.

See also

- “System procedures to add or delete scripts” on page 691
- “Adding and deleting scripts” on page 285
- “ml_add_dnet_connection_script system procedure” on page 695
- “ml_add_connection_script system procedure” on page 694
- “ml_add_table_script system procedure” on page 708
- “ml_add_java_connection_script system procedure” on page 698
- “Methods” on page 593
- “Writing synchronization scripts in .NET” on page 588

Example

The following example assigns the empDownloadCursor method of the EgClass class to the download_cursor event for the table emp.

```
call ml_add_dnet_table_script( 'ver1', 'emp',
'download_cursor','EgPackage.EgClass.empDownloadCursor' )
```

ml_add_java_connection_script system procedure

Use this system procedure to register or unregister a Java method as the script for a connection event.

Syntax

ml_add_java_connection_script (
'*version*',
'*event*',
'*script*'
)

Parameters

Syntax	Description
version	VARCHAR(128). The version name.
event	VARCHAR(128). The event name.
script	TEXT. The script contents. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For Oracle, this parameter is CLOB.

Remarks

To unregister a method, set the script contents parameter to null.

The script value is a public method in a class in the MobiLink server classpath (for example, MyClass.MyMethod).

When you ml_add_java_connection_script, the method is associated with the event and script version that you specify. If the version name is new, it is automatically inserted into the ml_version table.

See also

- “System procedures to add or delete scripts” on page 691
- “Adding and deleting scripts” on page 285
- “ml_add_connection_script system procedure” on page 694
- “ml_add_table_script system procedure” on page 708
- “ml_add_dnet_connection_script system procedure” on page 695
- “ml_add_dnet_table_script system procedure” on page 697
- “ml_add_java_table_script system procedure” on page 699
- “Methods” on page 527
- “Writing synchronization scripts in Java” on page 523

Example

The following example registers the endConnection method of the CustEmpScripts class for the end_connection event.

```
call ml_add_java_connection_script( 'ver1',
'end_connection',
'CustEmpScripts.endConnection' )
```

ml_add_java_table_script system procedure

Use this system procedure to register or unregister a Java method as the script for a table event.

Syntax

ml_add_java_table_script (
'*version*',
'*table*',
'*event*',
'*script*'
)

Parameters

Syntax	Description
version	VARCHAR(128). The version name.
table	VARCHAR(128). table name.
event	VARCHAR(128). The event name.
script	TEXT. The script content. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For Oracle, this parameter is CLOB.

Remarks

To unregister a method, set the script content parameter to null.

The *script* value is a public method in a class in the MobiLink server classpath (for example, MyClass.MyMethod).

When you call ml_add_java_table_script, the method is associated with the table, event, and script version that you specify. If the version name is new, it is automatically inserted into the ml_version table.

See also

- "System procedures to add or delete scripts" on page 691
- "Adding and deleting scripts" on page 285
- "ml_add_connection_script system procedure" on page 694
- "ml_add_table_script system procedure" on page 708
- "ml_add_dnet_connection_script system procedure" on page 695
- "ml_add_dnet_table_script system procedure" on page 697
- "ml_add_java_connection_script system procedure" on page 698
- "Methods" on page 527
- "Writing synchronization scripts in Java" on page 523

Example

The following example registers the empDownloadCursor method of the CustEmpScripts class for the download_cursor event for the table emp.

```
call ml_add_java_table_script( 'ver1', 'emp',
'download_cursor','CustEmpScripts.empDownloadCursor' )
```

ml_add_lang_connection_script system procedure

This procedure is for internal use only.

ml_add_lang_connection_script_chk system procedure

This procedure is for internal use only.

ml_add_lang_table_script system procedure

This procedure is for internal use only.

ml_add_lang_table_script_chk system procedure

This procedure is for internal use only.

ml_add_missing_dnld_scripts system procedure

Use this system procedure to define missing download_cursor and download_delete_cursor scripts as ignored scripts.

Syntax

ml_add_missing_dnld_scripts (
'*script_version_name***')**

Parameters

Syntax	Description
script_version_name	VARCHAR(128). The name of the script version.

See also

- "download_cursor table event" on page 347
- "download_delete_cursor table event" on page 349

ml_add_passthrough system procedure

Use this system procedure to identify remote databases that should execute a script. This procedure adds an entry to the ml_passthrough system table. If an entry with the given remote_id and run_order already exists in the table, this procedure updates the entry.

Syntax

ml_add_passthrough (
'*remote_id***',**
'*script_name***',**
run_order
)

Parameters

Syntax	Description
remote_id	VARCHAR(128). The remote ID of the database that should execute the script. This value can be a valid remote ID in the ml_database table to apply to a specific client, or null to apply to all the script clients listed in the ml_database table. **Caution** Be very careful when applying a script to all, or even many, remotes. A poorly written script can leave most or even all of your remotes damaged or disabled.
script_name	VARCHAR(128). The name of the script being subscribed to. This value must be a valid script name defined in the ml_passthrough_script table.

Syntax	Description
run_order	INTEGER. The run_order parameter determines the order in which scripts are applied on the remote database. Scripts are always applied in order by run_order. Each remote stores the run_order of the last script that it attempted to apply and does not download or execute any script with a run_order less than this. This value must be a non-negative integer or null.

Remarks

If you define run_order as null, the procedure assigns an integer based on the value of remote_id. If remote_id is null, the procedure assigns a value equal to the run_order value in ml_passthrough, plus 10. If remote_id is not null, the procedure assigns the maximum value of the run_order column for the remote_id in the ml_passthrough table plus 10.

ml_add_passthrough_repair system procedure

Use this system procedure to define rules for handling script errors. Each rule defines the action that a client should perform when a specific script generates a given error code. This procedure adds an entry to the ml_passthrough_repair system table. If an entry with the given failed_script_name and error_code already exists in the table, the procedure updates the entry.

Syntax

ml_add_passthrough_repair (
'*failed_script_name*',
error_code,
'*new_script_name*',
'*action*'
)

Parameters

Syntax	Description
failed_script_name	VARCHAR(128). The name of the failed script to which this rule applies. This value must be a valid script name in the ml_passthrough_script table.
error_code	INTEGER. The SQL Anywhere error code that this rule handles.
new_script_name	VARCHAR(128). The name of a script to replace the failed script when action is R. If action is S, P, or H, this value must be null. If action is R, this value must be a valid script name in the ml_passthrough_script table, and can be the same as failed_script_name.

Syntax	Description
action	CHAR(1). The action that a client should perform when error_code is generated for failed_script_name. This value must be one of the following: • **R** (replace) Indicates that the failed script should be replaced with the one specified by **new script name** and an attempt should be made to run the new script. To rerun the failed script, choose **new script name** to be the same as **failed script name**. • **P** (purge) Indicates that the remote database should discard all the scripts that it has received and continue executing script normally after that. • **S** (skip) Indicates that the remote database should ignore the failed script and continue executing scripts as if the failed script had succeeded. • **H** (halt) Indicates that the remote should not execute any more scripts until it receives further instructions.

Remarks

You should make every effort to avoid failed SQL passthrough scripts by testing scripts thoroughly.

ml_add_passthrough_script system procedure

Use this system procedure to create a passthrough script. This procedure adds an entry to the ml_passthrough_script system table.

Syntax

ml_add_passthrough_script (
'*script_name*',
'*flags*',
'*affected_pubs*',
'*script*',
'*description*'
)

Parameters

Syntax	Description
script_name	VARCHAR(128). The script name. This value must be unique.

Syntax	Description
flags	VARCHAR(256). The value that tells clients how to run the script. This value can be null or contain a combination of the following keywords in a semicolon-delimited list: • **manual** Indicates that the script may only be run in manual execution mode. By default, all scripts can be run in either automatic or manual execution modes. • **exclusive** Indicates that the script may only be automatically executed at the end of a synchronization where exclusive locks were obtained on all tables being synchronized. This option is ignored if the affected_pubs value lists no publications. This option is only meaningful to SQL Anywhere remotes. • **schema_diff** Indicates that the script should be run in schema-diffing mode. In this mode, the database schema is altered to match the schema described in the script. For example, a create statement for an existing table is treated as an alter statement. This flag only applies to scripts run on UltraLite remotes. For example: `'manual;exclusive;schema_diff'`
affected_pubs	TEXT. A list of publications that must be synchronized before the script is run. An empty string or null indicates that no synchronization is required. This value is only meaningful for SQL Anywhere clients. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For Oracle, this parameter is CLOB.

Syntax	Description

script	TEXT. The contents of the passthrough script. This value cannot be null. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For Oracle, this parameter is CLOB.
	
	The script content must be non-null. For UltraLite remotes, the **script** content should be a collection of SQL statements separated by the word **go**. Note that the word **go** must appear on a separate line. For SQL Anywhere remotes, the **script** content can be any collection of SQL statements that are valid when enclosed by a **begin...end** block.
	
	Example of **script** content on a SQL Anywhere remote:
	
	`DECLARE val INTEGER;`
	`SELECT c1 INTO val FROM t1 WHERE pk = 5;`
	`IF val > 100 THEN`
	`  INSERT INTO t2 VALUES ('c1 is big');`
	`ENDIF`
	
	Example of **script** content on an UltraLite remote:
	
	`CREATE TABLE myScript (c1 INT NOT NULL PRIMARY KEY)`
	`GO`
	`INSERT INTO myScript VALUES (1)`
	`GO`

description	VARCHAR(2000). A comment or description of the script. This value may be null.

Remarks

This procedure generates an error if the specified script_name already exists in ml_passthrough_script.

ml_add_property system procedure

Use this system procedure to add or delete MobiLink properties. This system procedure changes rows in the ml_property system table.

Syntax

ml_add_property (
'*comp_name*',
'*prop_set_name*',
'*prop_name*',
'*prop_value*'
)

Parameters

Syntax	Description
comp_name	VARCHAR(128). The component name. To save properties by script version, set to ScriptVersion. For MobiLink server properties, set to MLS. For server-initiated synchronization properties, set to SIS.
prop_set_name	VARCHAR(128). The property set name. If the component name is ScriptVersion, then this parameter is the name of the script version. If the component name is MLS, then this parameter can be either ml_user_log_verbosity to specify verbosity for a MobiLink user, or ml_remote_id_log_verbosity to specify verbosity for a remote ID. If the component name is SIS, then this parameter is the name of the Notifier, gateway, or carrier that you are setting a property for.
prop_name	VARCHAR(128). The property name. If the component name is ScriptVersion, then this parameter is a property that you define. You can reference these properties using DBConnectionContext: getVersion and getProperties, or ServerContext: getPropertiesByVersion, getProperties, and getPropertySetNames. If the component name is MLS, then this property is either a MobiLink user name or remote ID that you define.
prop_value	TEXT. The property value. If the prop_set_name is ml_user_log_verbosity or ml_remote_id_log_verbosity, this must be a valid mlsrv -v option. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For Oracle, this parameter is CLOB. To delete a property, set to null.

Log verbosity for targeted MobiLink users and remote IDs

The MobiLink server can be set to use different log verbosity for a targeted MobiLink user or remote ID. The MobiLink server checks the ml_property table every five minutes and looks for verbose settings for a MobiLink user or remote ID. If verbose settings exist, then it uses the new setting to log output messages for the given MobiLink user or remote ID. This enables you to see the details for a specific user or remote ID without the need for high verbosity settings that would negatively impact the server farm, and without requiring a restart of each server in the farm.

To set maximum verbosity for a targeted MobiLink user, for example *ml_user1*, log into the consolidated database and issue the following SQL command:


```
call ml_add_property( 'MLS', 'ml_user_log_verbosity', 'ml_user1', '-v+' )
```

To set maximum verbosity for a targeted remote ID, for example *rid_1*, log into the consolidated database and issue the following SQL command:

```
call ml_add_property( 'MLS', 'ml_remote_id_log_versity', 'rid_1', '-v+' )
```

Note that *verbose_setting* must be a valid MobiLink server -v option. For example, to log row data and undefined table scripts, the verbose_setting can be -vru or vru. The MobiLink server will use this verbose setting for ml_user1 or rid_1 after 5 minutes. See "-v mlsrv12 option" on page 70.

To disable log verbosity for a MobiLink user, log into the consolidated database and issue the following SQL command:

```
call ml_add_property( 'MLS', 'ml_user_log_verbosity', 'ml_user', NULL )
```

To disable log verbosity for a MobiLink remote ID, log into the consolidated database and issue the following SQL command:

```
call ml_add_property( 'MLS', 'ml_remote_id_log_verbosity', 'rid_1', NULL )
```

The MobiLink server stops using the previous verbose setting for *ml_user* or *rid_1* after five minutes.

If **both** the **ml_user_log_verbosity** and **ml_remote_id_log_verbosity** are set for a given MobiLink user and remote ID, and if the MobiLink user name and remote ID in a synchronization are identical to the given targeted MobiLink user and remote ID, the MobiLink server will use the **ml_remote_id_log_verbosity** setting to log output messages.

Server-initiated synchronization

For server-initiated synchronization, the ml_add_property system procedure allows you to set properties for Notifiers, gateways, and carriers.

For example, to add the property `server=mailserver1` for an SMTP gateway called x:

```
ml_add_property( 'SIS','SMTP(x)','server','mailserver1' );
```

The verbosity property applies to all Notifiers and gateways, and so you cannot specify a particular property set name. To change the verbosity setting, leave the property set name blank:

```
ml_add_property( 'SIS','','verbosity',2 );
```

Script Version

For regular MobiLink synchronization, you can use this system procedure to associate properties with a script version. In this case, set the component_name to ScriptVersion. You can specify any properties, and use Java and .NET classes to access them.

For example, to associate an LDAP server with a script version called MyVersion:

```
ml_add_property( 'ScriptVersion','MyVersion','ldap-server','MyServer' )
```

See also

- "MobiLink server settings for server-initiated synchronization" [*MobiLink - Server-Initiated Synchronization*]
- "MobiLink server settings for server-initiated synchronization" [*MobiLink - Server-Initiated Synchronization*]
- Java API DBConnectionContext: "getProperties method" on page 539 and "getVersion method" on page 541
- .NET API DBConnectionContext: "GetProperties method" on page 617 and "GetVersion method" on page 619
- Java API ServerContext: "getPropertiesByVersion method" on page 563, "getProperties method" on page 563, "getPropertySetNames method" on page 564
- .NET ServerContext: "getPropertiesByVersion method" on page 649, "getProperties method" on page 648, "getPropertySetNames method" on page 649

ml_add_table_script system procedure

Use this system procedure to add or delete SQL table scripts in the consolidated database.

Syntax

ml_add_table_script (
'*version*',
'*table*',
'*event*',
'*script*'
)

Parameters

Syntax	Description
version	VARCHAR(128). The version name.
table	VARCHAR(128). The table name.
event	VARCHAR(128). The event name.
script	TEXT. The script contents. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For Oracle, this parameter is CLOB.

Remarks

To delete a table script, set the script contents parameter to null.

When you add a script, the script is inserted into the ml_script table and the appropriate references are defined to associate the script with the table, event and script version that you specify. If the version name is new, it is automatically inserted into the ml_version table.

See also

- "System procedures to add or delete scripts" on page 691
- "Adding and deleting scripts" on page 285
- "ml_add_connection_script system procedure" on page 694
- "ml_add_dnet_connection_script system procedure" on page 695
- "ml_add_dnet_table_script system procedure" on page 697
- "ml_add_java_connection_script system procedure" on page 698
- "ml_add_java_table_script system procedure" on page 699

Example

The following command adds a table script associated with the upload_insert event on the Customer table.

```
call ml_add_table_script( 'default', 'Customer', 'upload_insert',
   'INSERT INTO Customer( cust_id, name, rep_id, active )
      VALUES ( {ml r.cust_id}, {ml r.name}, {ml r.rep_id}, 1 )' )
```

ml_add_user system procedure

This procedure is for internal use only.

ml_delete_passthrough system procedure

This stored procedure removes the row(s) in the ml_passthrough table that cause the specified script to be downloaded to the specified remote with the specified run order. If the script is downloaded to the remote before it is deleted then it is not deleted from the remote and executes as usual.

Syntax

ml_delete_passthrough (
'*remote_id*',
'*script_name*',
'*run_order*'
)

Parameters

Syntax	Description
remote_id	VARCHAR(128). The remote ID. If **remote_id** is null then all rows in the ml_passthrough table for the specified script name and run order are removed.
script_name	VARCHAR(128). The script name.
run_order	INTEGER. The run order of the script applied on the remote database. If **run_order** is null then all rows for the specified **remote_id** and **script_name** are removed from the ml_passthrough table regardless of their run order.

Remarks

The MobiLink server does not automatically remove entries from the ml_passthrough table. You must use this procedure to remove outdated passthrough scripts.

ml_delete_passthrough_repair system procedure

Use this system procedure to delete a repair rule from the ml_passthrough_repair system table.

Syntax

ml_delete_passthrough_repair (
'*failed_script_name***',**
error_code
)

Parameters

Syntax	Description
failed_script_name	VARCHAR(128). The name of the script to which a rule applied.
error_code	INTEGER. The error code for which the rule applied.

Remarks

The MobiLink server does not automatically remove entries from the ml_passthrough_repair table. You must use this procedure to remove outdated passthrough repair scripts.

ml_delete_passthrough_script system procedure

Use this system procedure to delete a passthrough script from the ml_passthrough_script system table.

Syntax

ml_delete_passthrough_script (
'*script_name***'**
)

Parameters

Syntax	Description
script_name	VARCHAR(128). The name of the script to remove.

Remarks

Scripts can not be removed if they are referenced in the ml_passthrough or ml_passthrough_repair system tables.

The MobiLink server does not automatically remove entries from the ml_passthrough_script table. You must use this procedure to remove outdated passthrough scripts.

ml_delete_sync_state system procedure

Use this procedure to delete unused or unwanted synchronization states.

Syntax

ml_delete_sync_state (
'*user*'**,**
'*remote_id*'
)

Parameters

Syntax	Description
user	VARCHAR(128). The MobiLink user name.
remote_id	VARCHAR(128). The remote ID.

Remarks

These parameters can be null. If all the parameters are null, the procedure does nothing.

This stored procedure deletes all the rows from the ml_subscription table for the given MobiLink user name and remote ID. It also removes this remote ID from the ml_database table, if the remote ID is no longer referenced by any rows in the ml_subscription table.

If the remote ID is null and the MobiLink user name is not null, it removes all the rows that are referenced by the given MobiLink user name from the ml_subscription table and all the remote IDs from the ml_database table, if these remote IDs are no longer referenced by any rows in the ml_subscription table.

If the MobiLink user name is null and the remote ID is not null, the MobiLink user is not removed by this stored procedure, even if all the remote IDs have been deleted from the ml_database table and this user is no longer referenced by any rows in the ml_subscription table. If this MobiLink user needs to be deleted, you may delete it by issuing a command such as

```
delete from ml_user where name = 'user_name'
```

where *user_name* is the MobiLink user you want to delete.

Use this stored procedure with extreme caution because the MobiLink server automatically adds this remote ID in the ml_database and ml_subscription tables without checking its synchronization status the next time the MobiLink client requests synchronization for this remote ID. It may cause data inconsistency to delete synchronization states for a remote ID that did not have a successful synchronization in the last synchronization attempt.

This procedure removes all the rows from the ml_subscription table and the ml_database table for the given remote ID.

Example

The following example cleans up MobiLink system table information about remote databases with the remote ID remote_db_for_John for the MobiLink user John:

```
CALL ml_delete_sync_state( 'John', 'remote_db_for_John' )
```

ml_delete_sync_state_before system procedure

Use this procedure to clean up the MobiLink system tables when you have dropped remote databases.

Syntax

ml_delete_sync_state_before (
'*ts*'
)

Parameters

Syntax	Description
ts	TIMESTAMP. The datetime must appear in exactly the order specified in the consolidated database. If the datetime format in the consolidated database is set to 'yyyy/mm/dd hh:mm:ss.ssss', then the timestamp must appear in the order year, month, day, hour, minute, second, fraction of second.

Remarks

This stored procedure removes rows from MobiLink system tables that pertain to remote databases that are no longer being used. In particular, it does the following:

- Deletes all the rows from the ml_subscription system table that have both the last_upload_time and last_download_time earlier than the given timestamp.
- Removes remote IDs from the ml_database system table if the remote IDs are no longer referenced by any rows in the ml_subscription table.

You should not use this system procedure for a time period that is so recent that it may delete rows for remote databases that have not actually been deleted. If you do, the deletion of the rows in ml_subscription and ml_database could cause problems for remote databases that are in an "unknown state" caused by an unsuccessful upload; in that unknown state, the remote relies on the MobiLink system tables to resend data.

The timestamp provided to this procedure must have a correct date-time format because the procedure does not validate the date-time format of the parameter.

Example

The following example cleans up MobiLink system table information about remote databases that have not synchronized since January 10, 2004. It works for a SQL Anywhere consolidated database where the date-time format in the consolidated database is yyyy/mm/dd hh:mm:ss.ssss.


```
CALL ml_delete_sync_state_before( '2004/01/10 00:00:00' )
```

ml_delete_user system procedure

This procedure is for internal use only.

ml_ra_add_agent_id system procedure

Use this procedure to define a new remote agent in the consolidated database.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the new agent to be defined in the consolidated database.

Remarks

If an Agent connects to the MobiLink server without ml_ra_add_agent_id being called first, then that Agent is automatically added to the consolidated database. However, all properties for that agent are default values.

See also

- “ml_ra_manage_remote_db system procedure” on page 713
- “ml_ra_clone_agent_properties system procedure” on page 715
- “ml_ra_set_agent_property system procedure” on page 728

ml_ra_manage_remote_db system procedure

Use this procedure to add an agent-managed remote database.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the new agent to be defined in the consolidated database.
schema_name	VARCHAR(128). This is an IN parameter that indicates the type of remote database being created. This schema name must have been previously defined in the consolidated database using Sybase Central.

Syntax	Description
conn_str	VARCHAR(128). This is an IN parameter that specifies the database connection string used by the agent to connect to the remote database.

See also

- "ml_ra_add_agent_id system procedure" on page 713
- "ml_ra_clone_agent_properties system procedure" on page 715

ml_ra_assign_task system procedure

Use this procedure to assign a task to a specific remote agent.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the agent to assign the task to.
task_name	VARCHAR(128). This is an IN parameter that specifies the name of the specific task being assigned.

Remarks

Tasks must first be defined in Sybase Central using the MobiLink 12 plug-in before calling this system procedure.

Some tasks target a specific remote database. If this is the case, the agent must be managing a remote database of that type.

If a task has been previously assigned to an agent and then subsequently completed, the task can be assigned again. This makes the task active again, and it will run according to its schedule.

Tasks must first be defined using Sybase Central before calling the task_name parameter.

See also

- "ml_ra_cancel_task_instance system procedure" on page 715
- "ml_ra_notify_task system procedure" on page 727
- "ml_ra_cancel_notification system procedure" on page 714

ml_ra_cancel_notification system procedure

Use this procedure to cancel a server initiated remote task (SIRT) request that is no longer needed.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the agent responsible for the task you are canceling.
task_name	VARCHAR(128). This is an IN parameter that specifies the name of the specific task being canceled.

See also

- "ml_ra_notify_task system procedure" on page 727
- "ml_ra_assign_task system procedure" on page 714
- "ml_ra_delete_task system procedure" on page 717
- "ml_ra_cancel_task_instance system procedure" on page 715

ml_ra_cancel_task_instance system procedure

Use this procedure to cancel a remote task instance.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the agent responsible for the task you are canceling.
task_name	VARCHAR(128). This is an IN parameter that specifies the name of the specific task being canceled.

Remarks

Use this system procedure to cancel a task that is no longer needed.

The canceled task is reported as being in the **Cancel Pending** state until the agent completes any active runs of the task and confirms the canceled state through a synchronization of the agent database.

See also

- "ml_ra_cancel_notification system procedure" on page 714
- "ml_ra_notify_task system procedure" on page 727
- "ml_ra_assign_task system procedure" on page 714
- "ml_ra_delete_task system procedure" on page 717

ml_ra_clone_agent_properties system procedure

Use this procedure to set all remote agent properties at once.

Parameters

Syntax	Description
dst_agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the destination agent being created.
src_agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the agent being cloned to create the new agent.

Remarks

Use this system procedure to set all the properties for a specified agent at once. All the properties of an existing agent are copied to the new agent. Individual agent properties can be set more easily using Sybase Central.

Assigned tasks and managed remotes are not copied to the new agent.

See also

- “ml_ra_set_agent_property system procedure” on page 728
- “ml_ra_add_agent_id system procedure” on page 713
- “ml_ra_manage_remote_db system procedure” on page 713

ml_ra_delete_agent_id system procedure

Use this procedure to delete a defined agent from the consolidated database.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the agent being deleted.

Remarks

If you delete an agent that was managing remote databases, those remote databases will become unmanaged.

See also

- “ml_ra_delete_events_before system procedure” on page 716
- “ml_ra_delete_remote_id system procedure” on page 717
- “ml_ra_unmanage_remote_db system procedure” on page 729

ml_ra_delete_events_before system procedure

Use this procedure to delete events that are no longer needed from the consolidated database.

Parameters

Syntax	Description
delete_rows_older_than	TIMESTAMP. This is an IN parameter that specifies Events older than the specified value are deleted from the consolidated database.

Remarks

If your remote tasks return status frequently, then large numbers of events can build up in the consolidated database.

See also

- "ml_ra_get_latest_event_id system procedure" on page 722

ml_ra_delete_remote_id system procedure

Use this procedure to delete a remote database that is no longer needed from the consolidated database.

Parameters

Syntax	Description
remote_id	VARCHAR(128). This is an IN parameter that specifies the remote ID that corresponds to the remote database to be deleted.

Remarks

This procedure fails if tasks are still active for the specified remote_id. To force deletion, first delete the agent_id that is managing the remote.

See also

- "ml_ra_delete_agent_id system procedure" on page 716
- "ml_ra_delete_task system procedure" on page 717

ml_ra_delete_task system procedure

Use this procedure to delete a remote task from the consolidated database.

Parameters

Syntax	Description
task_name	VARCHAR(128). This is an IN parameter that specifies the name of the remote task to be deleted.

Remarks

This system procedure fails if there are still active task instances.

See also

- “ml_ra_delete_agent_id system procedure” on page 716
- “ml_ra_delete_events_before system procedure” on page 716
- “ml_ra_delete_remote_id system procedure” on page 717

ml_ra_get_agent_events system procedure

Use this procedure to query events.

Parameters

Syntax	Description
start_at_event_id	BIGINT. This is an IN parameter that specifies the ID of the event from which to start the query.
max_events_to_fetch	BIGINT. This is an IN parameter that specifies the maximum number of events to fetch.

Returns

Result	Description
event_id	BIGINT. A unique ID assigned to each event. The value is incremented by 1 for each new event.
event_class	VARCHAR(1). The event class. The class can be either **I** for information or **E** for error.

Result	Description
event_type	VARCHAR(8). The event types are listed below. • **ANEW** A new agent was defined in the consolidated database. This can occur by calling ml_ra_add_agent or if the agent is not pre-configured when the agent connects to the consolidated database for the first time. • **AFIRST** Occurs on the first synchronization of an agent. • **ADUP** A duplicate agent_id was found, meaning two or more agents are trying to use the same ID. The result_text for this is the remote_id of the blocked agent's agent database. • **ARESET** An agent rebuilt its agent database. Some task progress and results may have been lost. • **TB** A task has begun execution. • **TE** The task execution ended without a fatal error. • **TW** A task execution is waiting for a retry interval before continuing. • **TAC** Task execution ended because a command aborted. • **TAT** Task execution ended because it exceeded the maximum allowed running time. • **TAR** Task execution ended because it exceeded the maximum retry count. • **TFS** The task completed and will not run again because it was a run-once task that succeeded. • **TFF** The task completed and will not run again because it was a run-once task that failed. • **TFE** The task completed and will not run again because the schedule for the task has expired. • **TFC** The task completed and will not run again because the task was canceled by the server. • **CR - Command Result** The result_code and result_text are populated with values specific to the type of command. • **CE - Command Error** The result_code and result_text are populated with values specific to the type of command.
agent_id	VARCHAR(128). The ID of the agent that produced this event.

Result	Description
remote_id	VARCHAR(128). The ID of the remote that the event applies to. This is only set for task-related events that target a specific remote database.
task_name	VARCHAR(128). The name of the task. This is only set for task-related events.
command_number	INTEGER. The command number within a task that this event applies to. This is only set for command-specific events.
run_number	BIGINT. The unique number assigned to each run of a task. This is only set for task-specific events.
duration	INTEGER. The amount of time taken by the event. This is only set for command-specific events.
event_time	TIMESTAMP. The time the event took place. For most events the time is based on the clock of the machine the agent is executing on.
event_received	TIMESTAMP. The time the event was received by the server. This is always set from the clock of the consolidated database.
result_code	BIGINT. An event-specific BIGINT. For example, for a SQL query command result, the code would be the SQLCODE.
result_text	LONG VARCHAR. An event-specific LONG VARCHAR. For example, for a SQL query command result, this column would contain a *CSV* format of the result set.
p_crsr	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

Remarks

Alternatively you can use the ml_ra_get_task_results procedure, which only fetches events related to a specific run of a task. You can pass in a null @run_number to get the latest run of a task. See "ml_ra_get_task_results system procedure" on page 723.

One way to use this procedure is to use ml_ra_get_agent_events to wait for a task-end event (**TE**) then call ml_ra_get_task_results to get each of the command results that might need processing.

See also

- "ml_ra_get_task_results system procedure" on page 723

ml_ra_get_agent_ids system procedure

Use this procedure to get all the agents in the consolidated database.

Returns

Result	Description
agent_id	VARCHAR(128). The agent ID.
remote_id	VARCHAR(128). The remote ID of the agent database.
last_down-load_time	TIMESTAMP. The last download time.
last_upload_time	TIMESTAMP. The last upload time.
active_task_count	INTEGER. The number of active tasks.
description	VARCHAR(2048). Reserved for future use.
p_crsr	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

See also

- "ml_ra_get_remote_ids system procedure" on page 723

ml_ra_get_agent_properties system procedure

Use this procedure to see all the properties set for an agent.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the agent you are setting properties for.

Returns

Results	Description
property_name	VARCHAR(128). The property name.
property_value	VARCHAR(2048). The value of the property.
last_modified	TIMESTAMP. The time the property was last modified.
p_crsr	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

See also

- "ml_ra_get_agent_ids system procedure" on page 720
- "ml_ra_clone_agent_properties system procedure" on page 715

ml_ra_get_latest_event_id system procedure

Use this procedure to help determine how many new events there are.

Parameters

Syntax	Description
event_id	BIGINT. This is an OUT parameter that specifies the ID of the latest event.

Remarks

To determine how many new events there are, call the ml_ra_get_latest_event_id system procedure and subtract the last event_id you processed.

See also

- "ml_ra_get_agent_events system procedure" on page 718

ml_ra_get_orphan_taskdbs system procedure

Use this procedure to see a list of orphan agent databases, meaning an agent database that does not have a valid agent ID.

Returns

Results	Description
remote_id	VARCHAR(128). The remote ID.
orig_agent_id	VARCHAR(128). The ID of the original agent the agent database was associated with.
last_sync	TIMESTAMP. The time of the last synchronization.
p_crsr	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

Remarks

Orphaned databases can be the result of a number of problems in a synchronization system, such as creating duplicate agent ids on different computers or having two agent databases trying to use the same agent id.

The remote_id field has the computer name in it to aid in diagnosing problems.

See also

- "ml_ra_reassign_taskdb system procedure" on page 727

ml_ra_get_remote_ids system procedure

Use this procedure to get all the remote databases in the consolidated database, excluding the agent databases.

Parameters

None.

Returns

Results	Description
remote_id	VARCHAR(128). The remote ID of the remote database.
schema_name	VARCHAR(128). The type of the remote database.
agent_id	VARCHAR(128). The agent ID of the remote database.
agent_conn_str	VARCHAR(2048). The agent connection string.
last_down-load_time	TIMESTAMP. The last download time.
last_upload_time	TIMESTAMP. The last upload time.
description	VARCHAR(128). The description of the database.
p_crsr	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

See also

- "ml_ra_get_agent_properties system procedure" on page 721

ml_ra_get_task_results system procedure

Use this procedure to get events related to a specific run of a task.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to get results for.
task_name	VARCHAR(128). This is an IN parameter that specifies the name of the task you want to get results for.
run_number	INTEGER. This is an IN parameter that specifies the run number you want results for.

Returns

Result	Description
event_id	BIGINT. A unique ID assigned to each event. The value is incremented by 1 for each new event.
event_class	VARCHAR(1). The event class. The class can be either **I** for information or **E** for error.
event_type	VARCHAR(8). The event type.
agent_id	VARCHAR(128). The ID of the agent that produced this event.
remote_id	VARCHAR(128). The ID of the remote that the event applies to. This is only set for task-related events that target a specific remote database.
task_name	VARCHAR(128). The name of the task. This is only set for task-related events.
command_number	INTEGER. The command number within a task that this event applies to. This is only set for command-specific events.
run_number	BIGINT. The unique number assigned to each run of a task. This is only set for task-specific events.
duration	INTEGER. The amount of time taken by the event. This is only set for command-specific events.
event_time	TIMESTAMP. The time the event took place. For most events the time is based on the clock of the machine the agent is executing on.
event_received	TIMESTAMP. The time the event was received by the server. This is always set from the clock of the consolidated database.
result_code	BIGINT. An event-specific BIGINT. For example, for a SQL query command result, the code would be the SQLCODE.

Result	Description
result_text	LONG VARCHAR. An event-specific LONG VARCHAR. For example, for a SQL query command result, this column would contain a *CSV* format of the result set.
p_crsr	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

Remarks

This system procedure only fetches events related to a specific run of a task. It is an alternative to the ml_ra_get_agent_events system procedure.

You can pass in a null @run_number to get the latest run of a task.

See also

- "ml_ra_get_agent_events system procedure" on page 718

Example

One way to use this procedure would be to use ml_ra_get_agent_events to wait for a task-end event then call ml_ra_get_task_results to get each of the command results that might need processing.

ml_ra_get_task_status system procedure

Use this procedure to check the status of tasks.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to get status for.
task_name	VARCHAR(128). This is an IN parameter that specifies the name of the task you want to get status for.

Returns

Result	Description
agent_id	VARCHAR(128). The ID of the agent that produced this event.
remote_id	VARCHAR(128). The ID of the remote that the event applies to.
task_name	VARCHAR(128). The name of the task.

Result	Description
task_id	BIGINT. The task ID.
state	VARCHAR(4). The state of the deployed task. State can be one of the following: • **P** Pending. Awaiting confirmation that the agent has received the task. • **A** Active. The agent has the task and will run it when it is scheduled to run. • **S** Succeeded. The task is complete and will not run again unless it is reassigned. • **F** Failed. The task is complete and will not run again unless it is reassigned. • **CP** Cancel pending. Waiting for confirmation that the agent has canceled the task. • **C** Canceled. The task is complete and will not run again unless it is reassigned. • **E** Expired. The task is complete and will not run again unless it is reassigned.
reported_exec_count	BIGINT. The reported number of tasks that have been executed.
reported_error_count	BIGINT. The reported number of errors.
reported_attempt_count	BIGINT. The reported number of attempts to execute a task.
last_status_update	TIMESTAMP. The time the last status update was given.
last_success	TIMESTAMP. The time of the last successful task.
assignment_time	TIMESTAMP. The time the task was assigned.
p_crsr	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

Remarks

The @agent_id and @task_name parameters can be set to null in order to get the status for all agent_ids, all task_names or both.

The reported_attempt_count may be greater than the reported_exec_count. This means that the precondition on the task evaluated to false on an attempt and the task did not execute.

The success count can be computed by subtracting reported_error_count from reported_exec_count.

See also

- "ml_ra_get_task_results system procedure" on page 723

ml_ra_notify_agent_sync system procedure

Use this procedure to cause an agent to synchronize its state.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to synchronize.

Remarks

This system procedure sends new tasks to the specified agent, and causes the agent to send any results it has from the execution of tasks to the MobiLink server.

See also

- "ml_ra_get_task_results system procedure" on page 723

ml_ra_notify_task system procedure

Use this procedure to run a task using server initiated remote tasks (SIRT).

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to run the task.
task_name	VARCHAR(128). This is an IN parameter that specifies the name of the task you want the agent to execute.

See also

- "ml_ra_cancel_notification system procedure" on page 714
- "ml_ra_delete_task system procedure" on page 717

ml_ra_reassign_taskdb system procedure

Use this procedure to reassign an agent database in the situation where you have an orphan agent database.

Parameters

Syntax	Description
taskdb_remote_id	VARCHAR(128). This is an IN parameter that specifies the remote ID of the orphaned agent database.
new_agent_id	VARCHAR(128). This is an IN parameter that specifies the ID of the new agent you want to assign the orphaned agent database to.

Remarks

If there are two agent databases that both want to use the same agent_id, the system considers the first agent database as the valid one, and the second agent database is considered an orphan, meaning it does not have a valid agent_id associated with it.

See also

- "ml_ra_get_orphan_taskdbs system procedure" on page 722

ml_ra_set_agent_property system procedure

Use this procedure to set remote agent properties.

Parameters

Syntax	Description
agent_id	VARCHAR(128). This is an IN parameter that specifies the agent ID.
property_name	VARCHAR(128). This is an IN parameter that specifies the property name to be set.
property_value	VARCHAR(2048). This is an IN parameter that specifies the property value to be set.

Remarks

The agent supports the following properties:

- **mlstream** The MobiLink stream parameters, for example `tcpip(host=localhost)`.
- **max_taskdb_sync_interval** The longest time in seconds that the agent should wait between synchronizing its agent database.
- **lwp_freq** The time between lightweight polls.

See also

- "ml_ra_clone_agent_properties system procedure" on page 715

ml_ra_unmanage_remote_db system procedure

Use this procedure to keep a remote database defined, but sever the link between the remote database and a remote agent, so that the database is no longer managed by its agent.

Parameters

Syntax	Description
remote_id	VARCHAR(128). This is an IN parameter that specifies the remote ID to be severed.
schema_name	VARCHAR(128). This is an IN parameter that indicates the type of the remote database.

Remarks

This procedure fails if there are tasks assigned to the remote database.

If you want the remote database to be managed by a different agent, you can call the ml_ra_manage_remote_db procedure again with a new agent_id.

See also

- "ml_ra_manage_remote_db system procedure" on page 713

ml_reset_sync_state system procedure

Use this procedure to reset synchronization state information in MobiLink system tables.

Syntax

ml_reset_sync_state (
'*user*',
'*remote_id*'
)

Parameters

Syntax	Description
user	VARCHAR(128). The MobiLink user name.
remote_id	VARCHAR(128). The remote ID.

Remarks

The parameters can be null. If both parameters are null, this procedure does nothing.

This stored procedure sets the progress, last_upload_time, and last_download_time columns in the ml_subscription table to their default values for the given user_name and remote ID. The default value for

the progress is 0. The default value for the last_upload_time and last_download_time columns is '1900/01/01 00:00:00'.

If the remote ID is null and the MobiLink user name is not null, this procedure sets those columns to the default values for the rows in the ml_subscription table referenced by the given MobiLink user name. If the MobiLink user name is null and the remote ID is not null, it sets them to the default values for the rows in the ml_subscription table with the given remote ID.

Use this stored procedure with extreme caution. The MobiLink server does not do any synchronization status checking for this remote ID the next time the MobiLink client requests synchronization for this remote ID. It may cause data inconsistency to reset a remote ID that did not have a successful synchronization in the last synchronization.

ml_server_delete system procedure

This procedure is for internal use only.

ml_server_update system procedure

This procedure is for internal use only.

MobiLink utilities

Introduction to MobiLink utilities

There are three MobiLink server utilities:

- "MobiLink stop utility (mlstop)" on page 731
- "MobiLink user authentication utility (mluser)" on page 732
- "MobiLink replay utility (mlreplay)" on page 733

In addition, see:

- MobiLink client utilities: "MobiLink client utilities" [*MobiLink - Client Administration*]
- UltraLite utilities: "UltraLite utilities" [*UltraLite - Database Management and Reference*]
- Utilities for using TLS certificates: "Certificate utilities" [*SQL Anywhere Server - Database Administration*]
- Other SQL Anywhere utilities: "Database administration utilities" [*SQL Anywhere Server - Database Administration*]

MobiLink stop utility (mlstop)

Stops the MobiLink server on the local computer.

Syntax

mlstop [*options*] [*name*]

Option	Description
@data	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used. 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*].
-f	Forced shutdown. Use if a hard shutdown does not work.
-h	Hard shutdown. MobiLink stops all synchronizations and exits. Some remotes may report an error.
-q	Quiet mode. This suppresses the banner.
-t *time*	Soft shutdown, with a hard shutdown after the specified time. *time* is a number followed by D, H, M, or S (for days, hours, minutes and seconds). For example, `-t 10m` specifies that the server should be shut down in 10 minutes or when current synchronizations complete, whichever is sooner. D, H, M, and S are not case sensitive.
-w	Waits for the MobiLink server to shut down before returning from the command.
name	If the MobiLink server is started using the -zs option, it must be shut down by specifying the same server name. See "-zs mlsrv12 option" on page 82.

Remarks

By default (if none of -f, -h or -t are specified), mlstop does a soft shutdown.

- **Soft shutdown** the MobiLink server stops accepting new connections and exits when the current synchronizations are complete.

- **Hard shutdown** the MobiLink server stops all synchronizations and exits. Some remotes may report an error.

MobiLink user authentication utility (mluser)

Registers MobiLink users at the consolidated database. For SQL Anywhere remotes, the users must have previously been created at the remote databases with the CREATE SYNCHRONIZATION USER statement.

Syntax

mluser [*options*] **-c** "*connection-string*"
{ **-f** *file* | **-u** *user* [**-p** *password*] }

Option	Description
@data	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used. 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*;..."	Use this to supply database connection parameters. The connection string must give the utility permission to connect to the consolidated database using an ODBC data source. This parameter is required.
-d	Deletes the user name(s) specified by -f or -u.
-f *filename*	Reads the user names and passwords from the specified file. The file should be a text file containing one user name and password pair on each line, separated by white space. You must specify either -f or -u.
-fips	When set, mluser fails if FIPS support is not installed.
-o *filename*	Logs output messages to the specified file.
-ot *filename*	Truncate the log file and then append output messages to it. The default is to send output to the screen.
-p *password*	Password to associate with the user. This option can only be used with -u.

Option	Description
-pc *collation-id*	Supplies a database collation ID for character set conversion of the user name and password. This should be one of the SQL Anywhere collation labels such as those listed in "Supported and alternate collations" [*SQL Anywhere Server - Database Administration*]. This option is required when user names and passwords are read from a file that is encoded in a different character set than the default character set determined by locale.
-u *ml_username*	Specify the user name to add (or delete, if used with -d). Only one user can be specified on a single command line. This option is used with -p if passwords are being used. You must specify either -f or -u.
-v	Specifies verbose logging.

Remarks

Given a user/password pair, the mluser utility first attempts to add the user. If the user has already been added to the consolidated database, it attempts to update the password for that user.

There are alternative ways to register user names in the consolidated database:

- Use Sybase Central.
- Specify the -zu+ command line option with mlsrv12. In this case, any existing MobiLink users that have not been added to the consolidated database are added when they first synchronize.

The MobiLink user must already exist in a remote database. To add users at the remote, you have the following options:

- For SQL Anywhere remotes, set the name with CREATE SYNCHRONIZATION USER and synchronize with that user name.
- For UltraLite remotes, you can either use the user_name field of the ul_sync_info structure; or in Java, use the SetUserName() method of the ULSynchInfo class before synchronizing.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "-zu mlsrv12 option" on page 83
- "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "Transport-layer security" [*SQL Anywhere Server - Database Administration*]

MobiLink replay utility (mlreplay)

The mlreplay utility is a tool used to replay MobiLink protocol information that is recorded by the MobiLink server. Each recorded file is called a **recorded protocol file**. Everything received from the start of a connection until the end of that connection is recorded in a separate recorded protocol file. Each recorded protocol file is named *recorded_protocol_x.mlr* where *x* is the job ID. The MobiLink server -rp option is used to specify that the MobiLink server should record all MobiLink protocol it receives from its clients. See "-rp mlsrv12 option" on page 63.

In addition to the data sent to and from the MobiLink server, the recorded protocol file also contains timing information so mlreplay can replay the recorded protocol information exactly as it was originally performed. The timing information is also used to try to make the simulated client take the same amount of time as the original client.

By default, mlreplay plays back the recorded protocol file without any changes. However, you can customize the replay session using a number of different options. The simulated client information consists of the username, password, remote ID, and last download time. This information can be customized using the -u, -p, -r (or -rg), and -ldt options respectively.

The mlreplay utility can also replay a recorded protocol file concurrently using the -sci <*simulated client information file*> option, where the simulated client information file is a *.CSV* file that lists the username, password, remote ID, and last download time for each simulated client. The mlreplay utility launches a simulated client for each simulated client information line in the simulated client information file.

Further customizations can be made to the replay session using the Replay API.

An additional tool, mlgenreplayapi, is included with the mlreplay utility. The mlgenreplayapi tool reads a recorded protocol file and generates the **Replay API** for the schema in that file. The Replay API can be modified (only the code in *mlreplaycallbacks.cpp* needs to be modified) to customize the data uploaded to the MobiLink server during the replay session. The Replay API is then compiled into the **replay DLL**, which mlreplay uses to customize the replay session. The replay DLL and a simulated client information file cannot be used at the same time. A callback is included in the Replay API that can be used to give the simulated client information for each simulated client. The number of simulated clients to launch when using the replay DLL is specified to mlreplay using the -n command line option. See "MobiLink Replay C ++ callback methods" on page 683.

Syntax

mlreplay [*options*] [*name=value* [*name2=value2...*]] [[*dll_name*] *filename*]

Option	Description
-ap	Adjust the progress of synchronizations being replayed in a replay session so that the mlreplay utility does not cause progress offset mismatch warnings on the MobiLink server.
-f *time_scale_factor*	A multiplier evenly applied to recorded times.

Option	Description
-ldt *last_download_time*	Specify the last download time to send to the MobiLink server during the replay session. If the recorded protocol being replayed contains multiple synchronizations (this is possible if a persistent connection was recorded) only the first last download time is replaced; the rest will be replaced by the last download time the MobiLink server sends MLReplay during the replay session. Even if the -ldt option is not used, MLReplay replaces the last download time in all but the first synchronization with the last download time received from the MobiLink server during the replay session. A last download time can also be specified using the simulated client information file (when the -sci option is used) or by the IdentifySimulatedClient callback when a DLL is provided.
-n *number_of_simulated_clients*	The number of simulated clients to run. The minimum is 1.
-o *file*	Log output messages to the specified file.
-ot *file*	Truncate the file and log output messages to it.
-p *password*	Replace passwords with the given password.
-ping *seconds*	Ping a MobiLink server to determine whether or not the server is ready to receive synchronizations. By default, mlreplay pings the server for 60 seconds. If the -ping option is used, the following return codes are valid: • **-1** Indicates that an error occurred. • **0** Indicates that mlreplay was able to ping the server and the server is ready to receive synchronizations. • **1** Indicates that mlreplay tried to ping the server but got no response; therefore, the server is not ready to receive synchronizations.
-r *remote ID*	Replace remote IDs with the given remote ID. This option cannot be used with the -rg option.
-rg	Replace remote IDs with a GUID.

Option	Description
-sci *file*	Provides mlreplay with a list of user names, passwords, and remote IDs to use for replaying synchronizations. mlreplay creates a thread for each line in the file to replay the synchronization with that client information. The format of each line should be [user name],[password],[remote ID]. If any field is left blank, mlreplay uses the corresponding value in the recorded synchronization. Note that the -u, -p, and -r options cannot be used with this option. If -rg is used, any remote IDs left blank are replaced with a GUID rather than the remote ID in the recorded synchronization.
-u *user name*	Replace user names with the given user name.
-x *stream(opts)*	The protocol stream and stream options to use to connect to the MobiLink server.

Remarks

The amount of time the original synchronization took is part of what is recorded, so mlreplay can attempt to replay the synchronization in the same amount of time.

The mlreplay utility can replay a recorded protocol file multiple times concurrently.

Use the following MobiLink server options with the mlreplay utility:

- **-rp** Use this option to specify the directory from which synchronizations are recorded for playback with the mlreplay utility.
- **-rrp** Use this option to run the mlreplay utility when the MobiLink server starts.
- **-lsc** Use this option to specify the connection information for the local server so the mlreplay utility can connect to the server.

See also

- "MobiLink Replay C++ callback methods" on page 683
- "-rrp mlsrv12 option" on page 62
- "-rp mlsrv12 option" on page 63
- "-lsc mlsrv12 option" on page 51

MobiLink arbiter server utility (mlarbiter)

The MobiLink arbiter server ensures that only a single MobiLink server in a server farm is running as the primary server. This prevents redundant notifications in a server-initiated synchronization environment and preserves messages in a QAnywhere messaging environment.

The mlarbiter command is used to start the MobiLink arbiter server. This command is used in conjunction with the MobiLink server -ca option, which provides the MobiLink server with the host name of the arbiter.

Syntax

mlarbiter

Remarks

The MobiLink arbiter listens port 4953 by default.

If the MobiLink server is not able make a connection to the arbiter after the arbiter started, the MobiLink tries to establish the connection every 15 seconds, and displays periodic error messages.

If the arbiter connection is dropped after the MobiLink servers in the server farm elected a primary server, the primary server shuts down immediately and the secondary servers will try to re-establish the arbiter connection every 15 seconds. After a connection to the arbiter is established, the MobiLink servers will re-elect a primary server.

Example

The following example shows how to use the MobiLink arbiter server with a MobiLink server farm.

To use the MobiLink arbiter server

1. Start the MobiLink arbiter on a computer with the following command line.

   ```
   mlarbiter
   ```

2. Start the MobiLink servers with a command line similar to the following. The MobiLink servers can be started on the same computer as the arbiter, or on different computers.

   ```
   mlsrv12 -c parameter1 -lsc parameter2 -ca Host_1 -notifier
   ```

   In the above example, *parameter1* is the consolidated database connection parameter and *parameter2* is the local MobiLink server connection parameter. All the MobiLink servers in the same server farm must contain the same setting for the -ca option.

See also

- "-ca mlsrv12 option" on page 40
- "-lsc mlsrv12 option" on page 51
- "Architecture" [*MobiLink - Getting Started*]

MobiLink data mappings between remote and consolidated databases

Adaptive Server Enterprise data mapping

Mapping to Adaptive Server Enterprise consolidated data types

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to Adaptive Server Enterprise consolidated data types. For example, a column of type FLOAT on the remote database should be type REAL on the consolidated database.

Maximum column length (MCL) depends on the Adaptive Server Enterprise page size. If the page size is 2K the MCL is 1954; if the page size is 4K the MCL is 4002. For information about MCL, see the Adaptive Server Enterprise documentation.

SQL Anywhere or UltraLite data type	Adaptive Server Enterprise data type	Notes
BIGINT	NUMERIC(20)[1] or BIGINT[2]	
BIT	BIT	
BINARY(*n*=<MCL)	BINARY(*n*)	
BINARY(*n*>MCL)	IMAGE	
CHAR(*n*=<MCL)	VARCHAR(*n*)	
CHAR(*n*>MCL)	TEXT	On download, ensure the values are not too long.
DATE	DATE[3] or DATETIME[4]	For Adaptive Server Enterprise DATETIME, the year must be in the range 1753-9999. For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.

SQL Anywhere or UltraLite data type	Adaptive Server Enterprise data type	Notes
DATETIME	DATETIME	The Adaptive Server Enterprise DATETIME values are accurate to 1/300 second. The last digit of the fractional second is always 0, 3, or 6. Other digits are rounded to one of these three digits, so 0 and 1 round to 0; 2, 3, and 4 round to 3; 5, 6, 7, and 8 round to 6; and 9 rounds to 10. For download, SQL Anywhere keeps the original values from Adaptive Server Enterprise, but for upload, the values may not be exactly the original values. If DATETIME is used for a primary key, conflict resolution may fail. To successfully synchronize DATETIME, you should round the fractional second to 10 milliseconds. Also, the year must be in the range 1753-9999.
DECIMAL(*p*<39, *s*)	DECIMAL(*p*,*s*)	The precision of the Adaptive Server Enterprise NUMERIC can be from 1 to 38 digits (*p*<39).
DECIMAL(*p*>=39,*s*)		There is no corresponding data type in Adaptive Server Enterprise.
DOUBLE	DOUBLE PRECISION	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	IMAGE	
INTEGER	INTEGER	
LONG BINARY	IMAGE	
LONG NVARCHAR	UNITEXT	
LONG VARBIT	TEXT	
LONG VARCHAR	TEXT	
MONEY	MONEY	
NCHAR(*c*=<MCL)	UNIVARCHAR(*c*/2)	

SQL Anywhere or UltraLite data type	Adaptive Server Enterprise data type	Notes
NCHAR(*c*>MCL)	UNITEXT	On download, ensure the values are not too long.
NTEXT	UNITEXT	
NUMERIC(*p*<39,*s*)	NUMERIC(*p*,*s*)	The precision of the Adaptive Server Enterprise decimal can be from 1 to 38 digits (*p*<39).
NUMERIC(*p*>=39,*s*)		
NVARCHAR(*c*=<MCL)	UNIVARCHAR(*c*/2)	
NVARCHAR(*c*>MCL)	UNITEXT	On download, ensure the values are not too long.
REAL	REAL	
SMALLDATETIME	DATETIME[4]	SQL Anywhere and UltraLite SMALLDATETIME is implemented as TIMESTAMP. The Adaptive Server Enterprise DATETIME is accurate to the minute. 29.998 seconds or lower are rounded down to the nearest minute; values with 29.999 seconds or higher are rounded up to the nearest minute. SQL Anywhere or UltraLite SMALLDATETIME is accurate to the microsecond. To successfully synchronize, SQL Anywhere or UltraLite SMALLDATETIME must be rounded to the minute. Also, the year must be in the range 1753-9999.
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	TEXT	

SQL Anywhere or UltraLite data type	Adaptive Server Enterprise data type	Notes
TIME	TIME[3] or DATETIME[4]	The Adaptive Server Enterprise TIME values are accurate to 1/300 second. The last digit of the fractional second is always 0, 3, or 6. Other digits are rounded to one of these three digits, so 0 and 1 round to 0; 2, 3, and 4 round to 3; 5, 6, 7, and 8 round to 6; and 9 rounds to 10. For download, SQL Anywhere keeps the original values from Adaptive Server Enterprise, but for upload, the values may not be exactly the original values. If TIME is used for a primary key, conflict resolution may fail. To successfully synchronize TIME, you should round the fractional second to 10 milliseconds.
TIMESTAMP	DATETIME	The Adaptive Server Enterprise DATETIME values are accurate to 1/300 second. The last digit of the fractional second is always 0, 3, or 6. Other digits are rounded to one of these three digits, so, 0 and 1 round to 0; 2, 3, and 4 round to 3; 5, 6, 7, and 8 round to 6; and 9 rounds to 10. For download, SQL Anywhere keeps the original values from Adaptive Server Enterprise, but for upload, the values may not be exactly the original values. If DATETIME is used for a primary key, conflict resolution may fail. To successfully synchronize DATETIME, you should round the fractional second to 10 milliseconds. Also, the year must be in the range 1753-9999.

SQL Anywhere or UltraLite data type	Adaptive Server Enterprise data type	Notes
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	There is no equivalent data type in Adaptive Server Enterprise. Therefore, a TIMESTAMP WITH TIME ZONE column should be mapped to a VARCHAR(34) column. In upload, the MobiLink server first converts the data to a string using the format YYYY-MM-DD HH:NN:SS.SSSSSS [+|-]HH:NN and then applies it to the consolidated database. In download, it converts the data from string to TIMESTAMP WITH TIME ZONE. Ensure the data in the consolidated database follows this format or the download will fail.
TINYINT	TINYINT	
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	Do not use UNIQUEIDENTIFIERSTR. Use UNIQUEIDENTIFIER instead.
UNSIGNED BIGINT	NUMERIC(20)[1] or UNSIGNED BIGINT[2]	
UNSIGNED INTEGER	UNSIGNED INT	
UNSIGNED SMALLINT	UNSIGNED SMALLINT	
UNSIGNED TINYINT	TINYINT	
VARBINARY(*n*=<MCL)	VARBINARY	
VARBINARY(*n*>MCL)	IMAGE	
VARBIT(*n*=<MCL)	VARCHAR(*n*)	
VARBIT(*n*>MCL)	TEXT	
VARCHAR(*n*=<MCL)	VARCHAR(*n*)	
VARCHAR(*n*>MCL)	TEXT	
XML	TEXT	

[1] Only applies to Adaptive Server Enterprise before version 15.0.

[2] Only applies to Adaptive Server Enterprise version 15.0 or later.

[3] Only applies to Adaptive Server Enterprise version 12.5.1 or later.

[4] Only applies to Adaptive Server Enterprise before version 12.5.1.

Mapping to SQL Anywhere or UltraLite remote data types

The following table identifies how Adaptive Server Enterprise consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type DOUBLE PRECISION on the consolidated database should be type DOUBLE on the remote database.

Adaptive Server Enterprise data type	SQL Anywhere or UltraLite data type	Notes
BIGINT[1]	BIGINT	
BINARY(*n*)	BINARY(*n*)	
BIT	BIT	
CHAR(*n*)	VARCHAR(*n*)	There is no equivalence between SQL Anywhere CHAR/NCHAR and Adaptive Server Enterprise CHAR/NCHAR. SQL Anywhere CHAR/NCHAR is equivalent to VARCHAR/NVARCHAR. You should not use CHAR/NCHAR in a consolidated database column that is synchronized. If you must use non-SQL Anywhere CHAR/NCHAR, run the MobiLink server with the -b option.
DATE	DATE	For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.

Adaptive Server Enterprise data type	SQL Anywhere or UltraLite data type	Notes
DATETIME	DATETIME	The Adaptive Server Enterprise DATETIME values are accurate to 1/300 second. The last digit of the fractional second is always one of 0, 3, or 6. Other digit numbers are rounded to one of these three digits, so 0 and 1 round to 0; 2, 3, and 4 round to 3; 5, 6, 7, and 8 round to 6; and 9 rounds to 10. For download, SQL Anywhere keeps the original values from Adaptive Server Enterprise, but for upload, the values may not be exactly the original values. Conflict resolution may fail. To successfully synchronize DATETIME, you should round the fractional second to 10 milliseconds. Also, the year must be in the range 1753-9999.
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	
DOUBLE PRECISION	DOUBLE	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	LONG BINARY	
INT	INT	
MONEY	MONEY	
NCHAR(*n*)	VARCHAR(*n*)	The Adaptive Server Enterprise NCHAR and NVARCHAR store multibyte national character strings, they are different from SQL Anywhere NCHAR and NVARCHAR. In a multibyte environment, use SQL Anywhere or UltraLite VARCHAR.
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	
NVARCHAR(*n*)	VARCHAR(*n*)	The Adaptive Server Enterprise NCHAR and NVARCHAR store multibyte national character strings, they are different from SQL Anywhere NCHAR and NVARCHAR. In a multibyte environment, use SQL Anywhere or UltraLite VARCHAR.

Adaptive Server Enterprise data type	SQL Anywhere or UltraLite data type	Notes
REAL	REAL	
SMALLDATETIME	SMALLDATETIME	SQL Anywhere and UltraLite SMALLDATETIME is implemented as TIMESTAMP. The Adaptive Server Enterprise SMALLDATETIME is accurate to the minute. 29.998 seconds or lower are rounded down to the nearest minute; values with 29.999 seconds or higher are rounded up to the nearest minute. SQL Anywhere or UltraLite SMALLDATETIME is accurate to the microsecond. To successfully synchronize, SQL Anywhere or UltraLite SMALLDATETIME must be rounded to the minute. Also, the year must be in the range 1900-2078.
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	LONG VARCHAR	
TIME	TIME	The Adaptive Server Enterprise TIME values are accurate to 1/300 second. The last digit of the fractional second is always one of 0, 3, or 6. Other digit numbers are rounded to one of these three digits, so 0 and 1 round to 0; 2, 3, and 4 round to 3; 5, 6, 7, and 8 round to 6; and 9 rounds to 10. For download, SQL Anywhere keeps the original values from Adaptive Server Enterprise, but for upload, the values may not be exactly the original values. Conflict resolution may fail. To successfully synchronize TIME, it is recommended that you round the fractional second to 10 milliseconds.

Adaptive Server Enterprise data type	SQL Anywhere or UltraLite data type	Notes
TIMESTAMP	VARBINARY(8)	Within Adaptive Server Enterprise, TIMESTAMP is a binary counter that gets incremented with every change to a row. So, each table can only contain one TIMESTAMP column and it does not make sense to synchronize it. If it must be a in synchronization, map it to a VARBINARY(8) data type in SQL Anywhere or UltraLite. This TIMESTAMP column cannot be explicitly inserted or updated, because it is maintained by the server. Keep this in mind when you are implementing upload scripts for tables that contain such columns.
TINYINT	TINYINT	
UNSIGNED BIGINT[1]	UNSIGNED BIGINT	
UNSIGNED INT[1]	UNSIGNED INT	
UNSIGNED SMALLINT[1]	UNSIGNED SMALLINT	
VARBINARY(*n*)	VARBINARY(*n*)	
VARCHAR(*n*)	VARCHAR(*n*)	
UNICHAR(*n*)	NVARCHAR(*n*)	Not available in UltraLite.
UNITEXT[1]	LONG NVARCHAR	Not available in UltraLite.
UNIVARCHAR(*n*)	NVARCHAR(*n*)	Not available in UltraLite.

[1] Only applies to Adaptive Server Enterprise before version 15.0.

IBM DB2 LUW data mapping

Mapping to IBM DB2 LUW consolidated data types

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to IBM DB2 LUW consolidated data types. For example, a column of type BIT on the remote database should be type SMALLINT on the consolidated database.

When creating an IBM DB2 LUW table, you need to pay attention to the DB2 page size. IBM DB2 LUW has a maximum row length (MRL) based on the page size: the MRL is 4005 when the page size is 4K, 8101 when 8K, 16293 when 16K and 32677 when 32K. The length of all columns in a table cannot exceed the above limitation. If a table has a BLOB or CLOB column, you count row length using the LOB locator, not BLOB or CLOB data directly. For details, see the IBM DB2 LUW documentation.

SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
BIGINT	BIGINT	
BINARY(*n*<MRL)	VARCHAR(*n*) FOR BIT DATA	
BINARY(*n*>=MRL)	BLOB(*n*)	
BIT	SMALLINT	
CHAR(*n*<MRL)	VARCHAR(*n*)	
CHAR(*n*>=MRL)	CLOB(*n*)	IBM DB2 LUW values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
DATE	DATE	For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.
DATETIME	TIMESTAMP	
DECIMAL(*p*<32,*s*)	DECIMAL(*p*,*s*)	The precision of SQL Anywhere DECIMAL is between 1 and 127. The maximum precision of IBM DB2 LUW DECIMAL is 31.
DECIMAL(*p*>=32,*s*)		Any data of SQL Anywhere DECIMAL precision greater than 31 cannot be synchronized to IBM DB2 LUW.
DOUBLE	DOUBLE	DOUBLE is an imprecise numeric data type that is subject to rounding. When working with different types of computers, the underlying storage of DOUBLE is often different, resulting in different rounding. DOUBLE is a bad choice to use in a primary key because primary keys are looking for equality. This is especially true in a synchronization environment because the consolidated database often runs on different hardware from the remote database.

SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
FLOAT(1-24)	REAL	FLOAT can cause problems if the consolidated and remote databases don't allow the exact same (imprecise) values. iAnywhere Solutions does not test all possible values, so care must be taken. To avoid problems, do not use these types as part of a primary key.
FLOAT(25-53)	DOUBLE	FLOAT can cause problems if the consolidated and remote databases don't allow the exact same (imprecise) values. iAnywhere Solutions does not test all possible values, so care must be taken. To avoid problems, do not use these types as part of a primary key.
IMAGE	BLOB(*n*)	
INTEGER	INTEGER	
LONG BINARY	BLOB(*n*)	
LONG NVARCHAR	CLOB(*n*)	There is no corresponding data type in IBM DB2 LUW. If the IBM DB2 LUW character set is Unicode, SQL Anywhere LONG NVARCHAR can synchronize to IBM DB2 CLOB. UltraLite doesn't have LONG NVARCHAR.
LONG VARBIT	CLOB(*n*)	
LONG VARCHAR	CLOB(*n*)	
MONEY	DECIMAL(19,4)	

SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
NCHAR(*c*)	VARCHAR(*n*) or CLOB(*n*)	There is no corresponding data type in IBM DB2 LUW. If the IBM DB2 LUW character set is Unicode, NCHAR can synchronize to IBM DB2 LUW VARCHAR or CLOB. The size of SQL Anywhere NCHAR is characters and the size of IBM DB2 LUW VARCHAR is bytes. If you map to VARCHAR, the total bytes of NCHAR can not be bigger than MRL. Otherwise, NCHAR should map to CLOB. It is difficult to calculate the number of bytes in NCHAR(*c*), but it is approximately *c*=*n*/4. In general, if *c* is less than MRL/4, map to VARCHAR(*n*), but if *c* is greater than or equal to MRL/4, map to CLOB(*n*).
NUMERIC(*p*<32,*s*)	NUMERIC(*p*,*s*)	
NUMERIC(*p*>=32,*s*)		There is no corresponding data type in IBM DB2 LUW.
NTEXT	CLOB(*n*)	There is no corresponding data type in IBM DB2 LUW. If the IBM DB2 LUW character set is Unicode, NTEXT can synchronize to IBM DB2 LUW CLOB.
NVARCHAR(*c*)	VARCHAR(*n*) or CLOB(*n*)	There is no corresponding data type in IBM DB2 LUW. If the IBM DB2 LUW character set is Unicode, NVARCHAR can synchronize to IBM DB2 LUW VARCHAR or CLOB. The size of SQL Anywhere NVARCHAR is characters and the size of IBM DB2 LUW VARCHAR is bytes. If you map to VARCHAR, the total bytes of NVARCHAR can not be bigger than MRL. Otherwise, NVARCHAR should map to CLOB. It is difficult to calculate the number of bytes in NVARCHAR(*c*), but it is approximately *c*=*n*/4. In general, if *c* is less than MRL/4, map to VARCHAR(*n*), but if *c* is greater than or equal to MRL/4, map to CLOB(*n*).

SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
REAL	REAL	REAL can cause problems if the consolidated and remote databases don't allow the exact same (imprecise) values. iAnywhere Solutions does not test all possible values, so care must be taken. To avoid problems, do not use these types as part of a primary key.
SMALLDATETIME	TIMESTAMP	
SMALLINT	SMALLINT	
SMALLMONEY	DECIMAL(10,4)	
TEXT	CLOB(*n*)	
TIME	TIMESTAMP or TIME	SQL Anywhere and UltraLite TIME values with fractional seconds require IBM DB2 LUW TIMESTAMP. SQL Anywhere and UltraLite time values with fractional seconds that are always zero can use IBM DB2 TIME. In order to preserve the precision of a time column, the MobiLink server always binds the TIME column with the ODBC SQL_TYPE_TIMESTAMP data type. When the consolidated database is running on a DB2 9.7 server, you may need to use DB2 conversion functions to explicitly convert the column between TIMESTAMP and TIME if the column is a part of a primary key.
TIMESTAMP	TIMESTAMP	
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	There is no equivalent data type in IBM DB2 LUW. Therefore, a TIMESTAMP WITH TIME ZONE column should be mapped to a VARCHAR(34) column. In upload, the MobiLink server first converts the data to a string using the format YYYY-MM-DD HH:NN:SS.SSSSSS [+|-]HH:NN and then applies it to the consolidated database. In download, it converts the data from string to TIMESTAMP WITH TIME ZONE. Ensure the data in the consolidated database follows this format or the download will fail.

SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
TINYINT	SMALLINT	For download, IBM DB2 LUW values must be non-negative.
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR is not recommended for IBM DB2 LUW. Use UNIQUEIDENTIFIER instead.
UNSIGNED BIGINT	DECIMAL(20)	For download, IBM DB2 LUW values must be non-negative.
UNSIGNED INTEGER	DECIMAL(11)	For download, IBM DB2 LUW values must be non-negative.
UNSIGNED SMALLINT	DECIMAL(5)	For download, IBM DB2 LUW values must be non-negative.
UNSIGNED TINYINT	SMALLINT	For download, IBM DB2 LUW values must be non-negative.
VARBINARY(*n*<MRL)	VARCHAR(*n*) FOR BIT DATA	
VARBINARY(*n*>=MRL)	BLOB(*n*)	
VARBIT(*n*<MRL)	VARCHAR(*n*)	
VARBIT(*n*>=MRL)	CLOB(*n*)	
VARCHAR(*n*<MRL)	VARCHAR(*n*)	
VARCHAR(*n*>=MRL)	CLOB(*n*)	IBM DB2 LUW values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
XML	CLOB(*n*)	

Mapping to SQL Anywhere or UltraLite remote data types

The following table identifies how IBM DB2 LUW consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type INT on the consolidated database should be type INTEGER on the remote database.

When creating an IBM DB2 table, you need to pay attention to the page size. IBM DB2 has a maximum row length based on the page size: the MRL is 4005 when the page size is 4K, 8101 when 8K, 16293 when 16K and 32677 when 32K. The length of all columns in a table cannot exceed the above limitation.

If a table has a BLOB or CLOB column, you count row length using the LOB locator, not BLOB or CLOB data directly. For details, see the IBM DB2 documentation.

IBM DB2 LUW data type	SQL Anywhere or UltraLite data type	Notes
BLOB	LONG BINARY	
BIGINT	BIGINT	
CHAR(*n*)	VARCHAR(*n*)	There is no equivalent to IBM DB2 LUW CHAR in SQL Anywhere. You should not use CHAR in a consolidated database column that is synchronized. If you must synchronize IBM DB2 LUW CHAR columns, run MobiLink server with the -b option.
CHAR(*n*) FOR BIT DATA	BINARY(*n*)	
CLOB(*n*)	LONG VARCHAR	
DATE	DATE	For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.
DBCLOB(*n*)	LONG VARCHAR	The data type DBCLOB(*n*) is only used for double-byte characters. SQL Anywhere does not have a corresponding data type. When the IBM DB2 LUW character set is Unicode, DBCLOB(*n*) is equivalent to CLOB.
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	
DOUBLE	DOUBLE	DOUBLE is an imprecise numeric data type that is subject to rounding. When working with different types of computers, the underlying storage of DOUBLE is often different, resulting in different rounding. DOUBLE is a bad choice to use in a primary key because primary keys are looking for equality. This is especially true in a synchronization environment because the consolidated database often runs on different hardware from the remote database.

IBM DB2 LUW data type	SQL Anywhere or UltraLite data type	Notes
FLOAT	DOUBLE	FLOAT can cause problems if the consolidated and remote databases don't allow the exact same (imprecise) values. iAnywhere Solutions does not test all possible values, so care must be taken. To avoid problems, do not use these types as part of a primary key.
GRAPHIC(*n*)	VARCHAR(2*n*)	IBM DB2 LUW GRAPHIC does blank-padding, but SQL Anywhere CHAR does not. It is recommended that you do not use this data type. The data type GRAPHIC is only used for double-byte characters. SQL Anywhere does not have a corresponding data type. When the IBM DB2 LUW character set is Unicode, GRAPHIC is equivalent to CHAR.
INT	INTEGER	
LONG VARCHAR	VARCHAR(32700)	
LONG VARCHAR FOR BIT DATA	VARBINARY(32700)	
LONG VARGRAPHIC(*n*)	VARCHAR(32700)	The data type LONG VARGRAPHIC is only used for double-byte characters. SQL Anywhere does not have a corresponding data type. When the IBM DB2 LUW character set is Unicode, LONG VARGRAPHIC is equivalent LONG VARCHAR.
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	
REAL	REAL	REAL can cause problems if the consolidated and remote databases don't allow the exact same (imprecise) values. iAnywhere Solutions does not test all possible values, so care must be taken. To avoid problems, do not use these types as part of a primary key.
SMALLINT	SMALLINT	

IBM DB2 LUW data type	SQL Anywhere or UltraLite data type	Notes
TIME	TIME	The fractional seconds values from SQL Anywhere TIME values are truncated on download. To avoid problems, do not use fractional seconds. In order to preserve the precision of a TIME column, the MobiLink server always binds the time column with the ODBC SQL_TYPE_TIMESTAMP data type. When the consolidated database is running on a DB2 9.7 server, you may need to use DB2 conversion functions to explicitly convert the column between TIMESTAMP and TIME if the column is a part of a primary key.
TIMESTAMP	TIMESTAMP	
VARCHAR(*n*)	VARCHAR(*n*)	
VARCHAR(*n*) FOR BIT DATA	VARBINARY(*n*)	
VARGRAPHIC(*n*)	VARCHAR(2*n*)	The data type VARGRAPHIC is only used for double-byte characters. SQL Anywhere does not have a corresponding data type. When the IBM DB2 LUW character set is Unicode, VARGRAPHIC is equivalent to VARCHAR.

Microsoft SQL Server data mapping

Mapping to Microsoft SQL Server consolidated data types

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to Microsoft SQL Server consolidated data types. For example, a column of type DATE on the remote database should be type DATETIME on the consolidated database.

SQL Anywhere or UltraLite data type	Microsoft SQL Server data type	Notes
BIGINT	BIGINT	
BINARY(*n*<=8000)	VARBINARY(*n*)	

SQL Anywhere or UltraLite data type	Microsoft SQL Server data type	Notes
BINARY(*n*>8000)	VARBINARY(MAX)	
BIT	BIT	
CHAR(*n*<=8000)	VARCHAR(*n*)	
CHAR(*n*>8000)	VARCHAR(MAX)	
DATE	DATE	
DATETIME	DATETIME2	Microsoft SQL Server DATETIME2 and TIME values are accurate to 100 nanoseconds. However, TIMESTAMP and TIME values are only accurate to 1 microsecond. To successfully synchronize DATETIME2 and TIME, it is recommended that you round the fractional second to 1 microsecond.
DECIMAL(*p*=<38,*s*)	DECIMAL(*p*,*s*)	Microsoft SQL Server DECIMAL/NUMERIC precision ranges from 1 to 38, so *p* must be less than 39.
DECIMAL(*p*>38,*s*)		There is no corresponding data type in Microsoft SQL Server.
DOUBLE	FLOAT(53)	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	VARBINARY(MAX)	
INTEGER	INT	
LONG BINARY	VARBINARY(MAX)	
LONG NVARCHAR	NVARCHAR(MAX)	
LONG VARBIT	VARCHAR(MAX)	
LONG VARCHAR	VARCHAR(MAX)	
MONEY	MONEY	

SQL Anywhere or UltraLite data type	Microsoft SQL Server data type	Notes
NCHAR(*n*<=4000)	NVARCHAR(*c*)	
NCHAR(*n*>4000)	NVARCHAR(MAX)	
NTEXT	NVARCHAR(MAX)	
NUMERIC(*p*=<38,*s*)	NUMERIC(*p*,*s*)	Microsoft SQL Server DECIMAL/NUMERIC precision ranges from 1 to 38, so *p* must be less than 39.
NUMERIC(*p*>38,*s*)		There is no corresponding data type in Microsoft SQL Server.
NVARCHAR(*n*<=4000)	NVARCHAR(*c*)	
NVARCHAR(*n*>4000)	NVARCHAR(MAX)	
REAL	REAL	
SMALLDATETIME	SMALLDATETIME	SQL Anywhere and UltraLite SMALLDATETIME is implemented as TIMESTAMP. Microsoft SQL Server SMALLDATETIME is accurate to the minute. 29.998 seconds or lower are rounded down to the nearest minute; values with 29.999 seconds or higher are rounded up to the nearest minute. SQL Anywhere or UltraLite SMALLDATETIME is accurate to the microsecond. To successfully synchronize, SQL Anywhere or UltraLite SMALLDATETIME must be rounded to the minute. The year must be in the range 1900-2078.
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	VARCHAR(MAX)	

SQL Anywhere or UltraLite data type	Microsoft SQL Server data type	Notes
TIME	TIME	Microsoft SQL Server DATETIME2 and TIME values are accurate to 100 nanoseconds. However, TIMESTAMP and TIME values are only accurate to 1 microsecond. To successfully synchronize DATETIME2 and TIME, it is recommended that you round the fractional second to 1 microsecond.
TIMESTAMP	DATETIME2	Microsoft SQL Server DATETIME2 and TIME values are accurate to 100 nanoseconds. However, TIMESTAMP and TIME values are only accurate to 1 microsecond. To successfully synchronize DATETIME2 and TIME, it is recommended that you round the fractional second to 1 microsecond.
TIMESTAMP WITH TIME ZONE	DATETIMEOFFSET	
TINYINT	TINYINT	For download, values must be non-negative.
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
UNIQUEIDENTIFIERSTR	UNIQUEIDENTIFIER	
UNSIGNED BIGINT	NUMERIC(20)	For download, values must be non-negative.
UNSIGNED INTEGER	NUMERIC(11)	For download, values must be non-negative.
UNSIGNED TINYINT	TINYINT	For download, values must be non-negative.
UNSIGNED SMALLINT	INT	For download, values must be non-negative.
VARBINARY(*n*<=8000)	VARBINARY(*n*)	
VARBINARY(*n*>8000)	VARBINARY(MAX)	
VARBIT(*n*<=8000)	VARCHAR(*n*)	
VARBIT(*n*>8000)	VARCHAR(MAX)	
VARCHAR(*n*<=8000)	VARCHAR(*c*)	

SQL Anywhere or UltraLite data type	Microsoft SQL Server data type	Notes
VARCHAR(*n*>8000)	VARCHAR(MAX)	
XML	XML or VARCHAR(MAX)	For Microsoft SQL Server 2005, use XML. For other versions, use VARCHAR(MAX).

Mapping to SQL Anywhere or UltraLite remote data types

The following table identifies how Microsoft SQL Server consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type TEXT on the remote database should be type LONG VARCHAR on the consolidated database.

Microsoft SQL Server data type	SQL Anywhere or UltraLite data type	Notes
BIGINT	BIGINT	
BINARY(*n*)	BINARY(*n*)	
BIT	BIT	
CHAR(*n*)	VARCHAR(*n*)	A Microsoft SQL Server CHAR column is blank padded. A SQL Anywhere CHAR column is not blank padded by default and is equivalent to a VARCHAR column. Therefore, try to avoid using the CHAR data type in the synchronization tables in Microsoft SQL Server. If you must use the CHAR data type in the Microsoft SQL Server consolidated database, run the MobiLink server with the -b command line option to help resolve the differences between SQL Anywhere CHAR and non-SQL Anywhere CHAR.
DATE	DATE	

Microsoft SQL Server data type	SQL Anywhere or UltraLite data type	Notes
DATETIME	TIMESTAMP or DATETIME	Microsoft SQL Server DATETIME values are accurate to 1/300 second. The last digit of the fractional second is always 0, 3, or 6. Other digits are rounded to one of these three digits, so, 0 and 1 round to 0; 2, 3, and 4 round to 3; 5, 6, 7, and 8 round to 6; and 9 rounds to 10. For download, SQL Anywhere keeps the original values from Microsoft SQL Server, but for upload, the values may not be exactly the original values. If DATETIME is used for a primary key, conflict resolution may fail. To successfully synchronize DATETIME, you should round the fractional second to 10 milliseconds. The year must be in the range 1753-9999.
DATETIME2	TIMESTAMP	Microsoft SQL Server DATETIME2 and TIME values are accurate to 100 nanoseconds. However, TIMESTAMP and TIME values are only accurate to 1 microsecond. To successfully synchronize DATETIME2 and TIME, it is recommended that you round the fractional second to 1 microsecond.
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	LONG BINARY	
INT	INT	
MONEY	MONEY	
NCHAR(*n*)	NVARCHAR(*c*)	Not available in UltraLite. There is no equivalence between SQL Anywhere NCHAR and non-SQL Anywhere NCHAR. SQL Anywhere NCHAR is equivalent to NVARCHAR. You should not use NCHAR in a consolidated database column that is synchronized. If you must use non-SQL Anywhere NCHAR, run the MobiLink server with the -b option.

Microsoft SQL Server data type	SQL Anywhere or UltraLite data type	Notes
NTEXT	LONG NVARCHAR	Not available in UltraLite.
NVARCHAR(*c*)	NVARCHAR(*c*)	Not available in UltraLite.
NVARCHAR(MAX)	LONG NVARCHAR	Not available in UltraLite.
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	
REAL	REAL	REAL can cause problems if the consolidated and remote databases don't allow the exact same (imprecise) values. iAnywhere Solutions does not test all possible values, so care must be taken. To avoid problems, do not use these types as part of a primary key.
SMALLDATETIME	SMALLDATETIME	SQL Anywhere and UltraLite SMALLDATETIME is implemented as TIMESTAMP. Microsoft SQL Server SMALLDATETIME is accurate to the minute. 29.998 seconds or lower are rounded down to the nearest minute; values with 29.999 seconds or higher are rounded up to the nearest minute. SQL Anywhere or UltraLite SMALLDATETIME is accurate to the microsecond. To successfully synchronize, SQL Anywhere or UltraLite SMALLDATETIME must be rounded to the minute. The year must be in the range 1900-2078.
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	LONG VARCHAR	
TIME	TIME	Microsoft SQL Server DATETIME2 and TIME values are accurate to 100 nanoseconds. However, TIMESTAMP and TIME values are only accurate to 1 microsecond. To successfully synchronize DATETIME2 and TIME, it is recommended that you round the fractional second to 1 microsecond.

Microsoft SQL Server data type	SQL Anywhere or UltraLite data type	Notes
TIMESTAMP	VARBINARY(8)	Within Microsoft SQL Server, TIMESTAMP is a binary counter that gets incremented with every change to a row. So, each table can only contain one TIMESTAMP column and it does not make sense to synchronize it. If it must be in a synchronization, map it to a VARBINARY(8) data type in SQL Anywhere or UltraLite. This timestamp column cannot be explicitly inserted or updated, because it is maintained by the server. Keep this in mind when you are implementing upload scripts for tables that contain such columns.
DATETIMEOFFSET	TIMESTAMP WITH TIME ZONE	
TINYINT	TINYINT	
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
VARBINARY(*n*)	VARBINARY(*n*)	
VARBINARY(MAX)	LONG BINARY	
VARCHAR(*n*)	VARCHAR(*n*)	
VARCHAR(MAX)	LONG VARCHAR	
XML	XML	

MySQL data mapping

Mapping to MySQL consolidated data types

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to MySQL consolidated data types. For example, a column of type TEXT on the remote database should be type LONGTEXT on the consolidated database.

SQL Anywhere or UltraLite data type	MySQL data type	Notes
BIGINT	BIGINT	
BINARY(*n*<=255)	BINARY(*n*)	
BINARY(*n*>255)	BLOB	
BIT	BIT	
CHAR(*n*<=255)	CHAR(*n*)	
CHAR(*n*>255)	TEXT(*n*)	
DATE	DATE	The year must range from 1000 to 9999.
DATETIME	DATETIME	The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.
DECIMAL(*p*<=65,*s*<=30)	DECIMAL(*p*,*s*)	
DECIMAL(*p*>65,*s*>30)		There is no corresponding data type in MySQL if the precision is greater than 65 or if the scale is greater than 30.
DOUBLE	DOUBLE	
FLOAT	FLOAT	
IMAGE	LONGBLOB	
INTEGER	INTEGER	
LONG BINARY	LONGBLOB	
LONG NVARCHAR	LONGTEXT CHARACTER SET UTF8	
LONG VARBIT	LONGTEXT	
LONG VARCHAR	LONGTEXT	
MONEY	NUMERIC(19,4)	
NCHAR(*n*<=255)	CHAR(*n*) CHARACTER SET UTF8	

SQL Anywhere or UltraLite data type	MySQL data type	Notes
NCHAR(*n*>255)	TEXT CHARACTER SET UTF8	
NTEXT	LONGTEXT CHARACTER SET UTF8	
NUMERIC(*p*<=65,*s*<=30)	DECIMAL(*p*,*s*)	
NUMERIC(*p*>65,*s*>30)		There is no corresponding data type in MySQL.
NVARCHAR(*n*)	VARCHAR(*n*) CHARACTER SET UTF8	
REAL	REAL	
SMALLDATETIME	DATETIME	The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.
SMALLINT	SMALLINT	
SMALLMONEY	NUMERIC(10,4)	
TEXT	LONGTEXT	
TIME	TIME	The MySQL TIME data type does not support fractional seconds.
TIMESTAMP	DATETIME	The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.

SQL Anywhere or UltraLite data type	MySQL data type	Notes
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	There is no equivalent data type in MySQL. Therefore, a TIMESTAMP WITH TIME ZONE column should be mapped to a VARCHAR(34) column. In upload, the MobiLink server first converts the data to a string using the format YYYY-MM-DD HH:NN:SS.SSSSSS [+|-]HH:NN and then applies it to the consolidated database. In download, it converts the data from string to TIMESTAMP WITH TIME ZONE. Ensure the data in the consolidated database follows this format or the download will fail.
TINYINT	TINYINT UNSIGNED	TINYINT is always unsigned in SQL Anywhere and UltraLite.
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	
VARBINARY(*n*)	VARCHAR(*n*)	
VARBIT(*n*<=8000)	VARCHAR(*n*)	
VARBIT(*n*>8000)	TEXT	
VARCHAR(*n*)	VARCHAR(*n*)	
XML	LONGTEXT	

Mapping to SQL Anywhere or UltraLite remote data types

The following table identifies how MySQL consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type BOOL on the consolidated database should be type BIT on the remote database.

MySQL data type	SQL Anywhere or UltraLite data type	Notes
BIGINT	BIGINT	
BINARY(*n*)	BINARY(*n*)	
BIT(1)	BIT	
BIT(*n*>1)	UNSIGNED BIGINT	

MySQL data type	SQL Anywhere or UltraLite data type	Notes
BLOB(*n*<=32767)	VARBINARY(*n*)	
BLOB(*n*>32767)	IMAGE	
BOOL	BIT	
CHAR(*n*)	CHAR(*n*)	
DATE	DATE	The year must range from 1000 to 9999.
DATETIME	DATETIME	The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.
DOUBLE	DOUBLE	
DECIMAL	DECIMAL	
ENUM		There is no corresponding data type in SQL Anywhere or UltraLite.
GEOMETRY		There is no corresponding data type in SQL Anywhere or UltraLite.
INTEGER	INTEGER	
LINESTRING		There is no corresponding data type in SQL Anywhere or UltraLite.
LONGBLOB	IMAGE	
LONGTEXT	TEXT	
MEDIUMBLOB	IMAGE	
MEDIUMINT	INTEGER	
MEDIUMTEXT	TEXT	
MULTILINESTRING		There is no corresponding data type in SQL Anywhere or UltraLite.
MULTIPOINT		There is no corresponding data type in SQL Anywhere or UltraLite.

MySQL data type	SQL Anywhere or UltraLite data type	Notes
MULTIPOLYGON		There is no corresponding data type in SQL Anywhere or UltraLite.
NCHAR	NCHAR	Not available in UltraLite.
NUMERIC	NUMERIC	
NVARCHAR	NVARCHAR	Not available in UltraLite.
POINT		There is no corresponding data type in SQL Anywhere or UltraLite.
POLYGON		There is no corresponding data type in SQL Anywhere or UltraLite.
REAL	REAL	
SET		There is no corresponding data type in SQL Anywhere or UltraLite.
SMALLINT	SMALLINT	
TEXT(*n*<=32767)	VARCHAR(*n*)	
TEXT(*n*>32767)	TEXT	
TIME	TIME	The MySQL TIME data type does not support fractional seconds. The range of TIME in MySQL is '-838:59:59' to '838:59:59'. The range of TIME in SQL Anywhere or UltraLite is '00:00:00.000000' to '23:59:59:999999'.
TIMESTAMP	TIMESTAMP	The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999. Although MySQL offers automatic initialization and updating on TIMESTAMP columns, SQL Anywhere and UltraLite only offers automatic initialization.
TINYBLOB	VARBINARY	
TINYINT	SMALLINT	TINYINT is always unsigned in SQL Anywhere and UltraLite. Must be a positive value.

MySQL data type	SQL Anywhere or UltraLite data type	Notes
TINYINT UNSIGNED	TINYINT	TINYINT is always unsigned in SQL Anywhere and UltraLite.
TINYTEXT	VARCHAR	
VARBINARY(*n*<=32767)	VARBINARY(*n*)	
VARBINARY(*n*>32767)	IMAGE	
VARCHAR(*n*<=32767)	VARCHAR(*n*)	
VARCHAR(*n*>32767)	TEXT	
YEAR[(2|4)]	INTEGER	SQL Anywhere and UltraLite do not support the YEAR data type. YEAR needs to be mapped to INTEGER in a remote database. The INTEGER value must range from 1000 to 9999.

Oracle data mapping

Mapping to Oracle consolidated data types

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to Oracle consolidated data types. For example, a column of type BIT on the remote database should be type NUMBER on the consolidated database.

SQL Anywhere or UltraLite data type	Oracle data type	Notes
BIGINT	NUMBER(20)	
BINARY(*n*<=2000)	RAW(*n*)	
BINARY(*n*>2000)	BLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
BIT	NUMBER(1)	

SQL Anywhere or UltraLite data type	Oracle data type	Notes
CHAR(*n*<=4000)	VARCHAR2(*n* byte)	Oracle VARCHAR2 allows you to specify the maximum number of bytes or characters. The maximum length of VARCHAR2 data is 4000 bytes. If you specify the character number, make sure the maximum data length is not over 4000 bytes.
CHAR(*n*>4000)	CLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
DATE	DATE[2] or TIMESTAMP	SQL Anywhere or UltraLite fractional seconds cannot be preserved when using an Oracle DATE data type which has no fractional seconds. To avoid problems, do not use fractional seconds. The year must be in the range 1-9999. When using the Interactive SQL utility, turn off the Return_date_time_as_string option before executing your SQL statement.
DATETIME	DATE[2] or TIMESTAMP	SQL Anywhere or UltraLite fractional seconds cannot be preserved when using an Oracle DATE data type which has no fractional seconds. To avoid problems, do not use fractional seconds. The year must be in the range 1-9999. When using the Interactive SQL utility, turn off the Return_date_time_as_string option before executing your SQL statement.
DECIMAL(*p*<=38,*s*)	NUMBER(*p*, 0<=*s*<=38)	In SQL Anywhere DECIMAL, p is between 1 and 127, and s is always less than or equal to p. In Oracle NUMBER, p ranges from 1 to 38, and s ranges from -84 to 127. To synchronize, the Oracle NUMBER scale must be restricted to between 0 and 38.
DECIMAL(*p*>38,*s*)		There is no corresponding data type in Oracle.

SQL Anywhere or UltraLite data type	Oracle data type	Notes
DOUBLE	DOUBLE PRECISION or BINARY_DOUBLE[1]	The special values INF, -INF and NAN of Oracle Database 10g BINARY_FLOAT and BINARY_DOUBLE cannot be synchronized with SQL Anywhere or UltraLite.
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	BLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
INTEGER	INT	
LONG BINARY	BLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
LONG NVARCHAR	NCLOB	Oracle CLOB and NCLOB can hold up to 4G of data. SQL Anywhere LONG VARCHAR and LONG NVARCHAR can only hold up to 2G. Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
LONG VARBIT	CLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
LONG VARCHAR	CLOB	Oracle CLOB and NCLOB can hold up to 4G of data. SQL Anywhere LONG VARCHAR and LONG NVARCHAR can only hold up to 2G. Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
MONEY	NUMBER(19,4)	

SQL Anywhere or UltraLite data type	Oracle data type	Notes
NCHAR(*c*)	NVARCHAR2(*c* char) or NCLOB	The size of SQL Anywhere NCHAR and Oracle NVARCHAR2 indicates the maximum number of Unicode characters. The data length of Oracle NVARCHAR2 cannot be over 4000 bytes. It is difficult to calculate the maximum byte length from character size. In general, if the size is over 1000, map to NCLOB, otherwise map to NVARCHAR2.
NTEXT	NCLOB	Oracle NCLOB can hold up to 4G of data. SQL Anywhere NTEXT (or LONG NVARCHAR) can only hold up to 2G. Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
NUMERIC(*p*<=38,*s*)	NUMBER(*p*, 0<=*s*<=38)	In SQL Anywhere NUMERIC, *p* is between 1 and 127, and *s* is always less than or equal to *p*. In Oracle NUMBER, *p* ranges from 1 to 38, and *s* ranges from -84 to 127. To synchronize, the Oracle NUMBER scale must be restricted to between 0 and 38.
NUMERIC(*p*>38,*s*)		There is no corresponding data type in Oracle.
NVARCHAR	NVARCHAR2(*c* char) or NCLOB	The size of SQL Anywhere NCHAR and Oracle NVARCHAR2 indicates the maximum number of Unicode characters. The data length of Oracle NVARCHAR2 cannot be over 4000 bytes. It is difficult to calculate the maximum byte length from character size. In general, if the size is over 1000, map to NCLOB, otherwise map to NVARCHAR2.
REAL	REAL or BINARY_FLOAT[1]	The special values INF, -INF and NAN of Oracle Database 10g BINARY_FLOAT and BINARY_DOUBLE cannot be synchronized with SQL Anywhere or UltraLite.

SQL Anywhere or UltraLite data type	Oracle data type	Notes
SMALLDATETIME	DATE[2] or TIMESTAMP	SQL Anywhere or UltraLite fractional seconds cannot be preserved when using an Oracle DATE data type which has no fractional seconds. To avoid problems, do not use fractional seconds. The year must be in the range 1-9999.
SMALLINT	NUMBER(5)	
SMALLMONEY	NUMBER(10,4)	
TEXT	CLOB	Oracle CLOB can hold up to 4G of data. SQL Anywhere TEXT (or LONG VARCHAR) can only hold up to 2G. Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
TIME	DATE[2] or TIMESTAMP	SQL Anywhere or UltraLite fractional seconds cannot be preserved when using an Oracle DATE data type which has no fractional seconds. To avoid problems, do not use fractional seconds. When using the Interactive SQL utility, turn off the Return_date_time_as_string option before executing your SQL statement.
TIMESTAMP	DATE[2] or TIMESTAMP	SQL Anywhere or UltraLite fractional seconds cannot be preserved when using an Oracle DATE data type which has no fractional seconds. To avoid problems, do not use fractional seconds. The year must be in the range 1-9999. When using the Interactive SQL utility, turn off the Return_date_time_as_string option before executing your SQL statement.
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH TIME ZONE	
TINYINT	NUMBER(3)	For download, Oracle values must be non-negative.

SQL Anywhere or UltraLite data type	Oracle data type	Notes
UNSIGNED BIGINT	NUMBER(20)	For download, Oracle values must be non-negative.
UNSIGNED INTEGER	NUMBER(11)	For download, Oracle values must be non-negative.
UNSIGNED SMALLINT	NUMBER(5)	For download, Oracle values must be non-negative.
UNSIGNED TINYINT	NUMBER(3)	For download, Oracle values must be non-negative.
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR is not recommended to use for Oracle. Use UNIQUEIDENTIFIER instead.
VARBINARY(*n*<=2000)	RAW(*n*)	
VARBINARY(*n*>2000)	BLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
VARBIT(*n*<=4000)	VARCHAR2(*n* byte)	
VARBIT(*n*>4000)	CLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
VARCHAR(*n*<=4000)	VARCHAR2(*n* byte)	Oracle VARCHAR2 allows you to specify the maximum number of bytes or characters. The maximum length of VARCHAR2 data is 4000 bytes. If you specify the character number, make sure the maximum data length is not over 4000 bytes.
VARCHAR(*n*>4000)	CLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.

SQL Anywhere or UltraLite data type	Oracle data type	Notes
XML	CLOB	Oracle CLOB and NCLOB can hold up to 4G of data. SQL Anywhere XML can only hold up to 2G. Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.

[1] Only applies to Oracle Database 10g or later.

[2] Only applies to Oracle version 8i.

Note
The LONG data types are deprecated in Oracle 8, 8i and 9i.

Mapping to SQL Anywhere or UltraLite remote data types

The following table identifies how Oracle consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type LONG on the consolidated database should be type LONG VARCHAR on the remote database.

Oracle data type	SQL Anywhere or UltraLite data type	Notes
BFILE	LONG BINARY	Download only. Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
BINARY_DOUBLE	DOUBLE	The special values INF, -INF and NAN of BINARY_FLOAT cannot be synchronized with SQL Anywhere or UltraLite. The precision of FLOAT and DOUBLE in Oracle is different from SQL Anywhere and UltraLite. The value of the data may change depending on the precision.
BINARY_FLOAT	REAL	The special values INF, -INF and NAN of BINARY_FLOAT cannot be synchronized with SQL Anywhere or UltraLite. The precision of FLOAT and DOUBLE in Oracle is different from SQL Anywhere and UltraLite. The value of the data may change depending on the precision.

Oracle data type	SQL Anywhere or UltraLite data type	Notes
BLOB	LONG BINARY	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
CHAR(*n* byte)	VARCHAR(*n*)	There is no equivalence between SQL Anywhere CHAR and Oracle CHAR. SQL Anywhere CHAR is equivalent to VARCHAR. You should not use CHAR/NCHAR in a consolidated database column that is synchronized. If you must use non-SQL Anywhere CHAR, run the MobiLink server with the -b option. SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.
CLOB	LONG VARCHAR	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
DATE	TIMESTAMP	The year must be in the range 1-9999.
INTERVAL YEAR(*year_precision*) TO MONTH		There is no corresponding data type in SQL Anywhere or UltraLite.
INTERVAL DAY(*day_precision*) TO SECOND(*p*)		There is no corresponding data type in SQL Anywhere or UltraLite.
LONG	LONG VARCHAR	
LONG RAW	LONG BINARY	
NCHAR(*c* char)	NVARCHAR(*c*)	There is no equivalence between SQL Anywhere NCHAR and Oracle NCHAR. SQL Anywhere NCHAR is equivalent to NVARCHAR. You should not use NCHAR in a consolidated database column that is synchronized. If you must use non-SQL Anywhere NCHAR, run the MobiLink server with the -b option. SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.

Oracle data type	SQL Anywhere or UltraLite data type	Notes
NCLOB	LONG NVARCHAR	Not available in UltraLite. Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
NUMBER(*p*,*s*)	NUMBER(*p*,*s*)	In SQL Anywhere NUMBER, *p* is between 1 and 127, and *s* is always less than or equal to *p*. In Oracle NUMBER, *p* ranges from 1 to 38, and *s* ranges from -84 to 127. To synchronize, the Oracle NUMBER scale must be between 0 and 38.
NVARCHAR2(*c* char)	NVARCHAR(*c*)	Not available in UltraLite. SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.
RAW	BINARY	SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.
ROWID	VARCHAR(64)	UROWID and ROWID are read-only and so are unlikely to be synchronized.
TIMESTAMP(*p*<=6)	TIMESTAMP	When *p*<6, you may need to ensure SQL Anywhere or UltraLite values have the same precision. Otherwise, conflict detection may fail and/or duplicate rows may result. The year must be in the range 1-9999.
TIMESTAMP(*p*>6)		
TIMESTAMP(*p*) WITH LOCAL TIME ZONE		There is no corresponding data type in SQL Anywhere or UltraLite.
TIMESTAMP(*p*<=6) WITH TIME ZONE	TIMESTAMP WITH TIME ZONE	
TIMESTAMP(*p*) WITH TIME ZONE		
UROWID	VARCHAR(64)	UROWID and ROWID are read-only and so are unlikely to be synchronized.

Oracle data type	SQL Anywhere or UltraLite data type	Notes
VARCHAR2(*n* byte)	VARCHAR(*n*)	SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.

Character set considerations

Character set considerations

Each character of text is represented in one or more bytes. The mapping from characters to binary codes is called the **character set encoding**. Some character sets used for languages with small alphabets, such as European languages, use a single-byte representation. Others, such as Unicode, use a double-byte representation. Because they use twice the storage space for each character, double-byte character sets can represent a much larger number of characters.

Conversion errors can occur or data can be lost when text using one character set must be converted to another character set. Not all characters can be represented in all character sets. In particular, single-byte character sets can represent a much smaller number of characters than multibyte systems because of the limited number of codes available.

When the character set of your MobiLink remote database is the same as your consolidated database, character conversion issues are avoided.

Text often needs to be sorted to build indexes and to prepare ordered result sets, such as directory listings. The **sort order** identifies the order of the characters. For example, a sort order typically states that the letter "a" comes before the letter "b", which comes before the letter "c".

Each database has a **collation sequence**. You set the collation sequence when you create the database, although how you do so can differ between database systems. The collation sequence defines both the character set and the sort order for that database.

Tip
Whenever possible, define the collation sequence of your remote database to be the same as that of your consolidated database. This arrangement reduces the chance of erroneous conversions.

See also

- SQL Anywhere clients: “International languages and character sets” [*SQL Anywhere Server - Database Administration*]
- UltraLite clients: “UltraLite character sets” [*UltraLite - Database Management and Reference*]
- Information specific to your RDBMS: “MobiLink consolidated databases” on page 1

Character set conversion during synchronization

During synchronization, characters may need to be converted from one character set to another. The following conversions occur as characters are passed between the remote application and the consolidated database.

Character set conversion during upload

The MobiLink client sends data to the MobiLink server using the character set of the remote database.

1. The MobiLink server communicates with the consolidated database using the Unicode ODBC API. To do so, the MobiLink server converts all characters received from the remote database into Unicode and sends the Unicode to the ODBC driver.

2. If necessary, the ODBC driver for the consolidated database server converts the characters from Unicode into the character set of your consolidated database. This conversion is controlled solely by the ODBC driver for your consolidated database system. So, behavior can differ between two different database systems, particularly systems made by different manufacturers. MobiLink synchronization works with several database systems. Check the documentation of your particular consolidated server and ODBC driver for details.

Character set conversion during download

1. The ODBC driver for the consolidated database system receives characters in the coding of the consolidated database. It converts these characters into Unicode to pass them through the Unicode API to the MobiLink server. This conversion is controlled solely by the ODBC driver for your consolidated database system. Check the documentation of your particular consolidated server and ODBC driver for details.

2. The MobiLink server receives characters through the Unicode ODBC API. If the remote database uses a different character set, the MobiLink server converts the characters before downloading them.

Examples

- UltraLite applications on Windows Mobile devices use the Unicode character set.

  When you synchronize a Windows Mobile application, no character conversion occurs within the MobiLink server. The server finds that data arriving from the application is already in Unicode and passes it directly to the ODBC driver. Similarly, no character set conversion is necessary when downloading data.

- All SQL Anywhere databases and all UltraLite applications on platforms other than Windows Mobile use the character set determined by the collating sequence of the remote database.

  When you synchronize a remote database, the MobiLink server performs character set conversions between the character set of the remote database and Unicode.

Controlling ODBC driver character set conversion

Because most consolidated databases are unlikely to use Unicode, it is important to understand how the ODBC driver for your consolidated database system converts data to and from Unicode. Some ODBC

drivers use the language settings of the computer running MobiLink to determine what character set to use. In these cases, it is best if the language and code-page settings of the computer running the MobiLink server match those of the consolidated database.

Other ODBC drivers, such as the driver for Sybase Adaptive Server Enterprise, allow each connection to use a specific character set. To avoid conversion errors, the character set used by MobiLink should be set to match that of the consolidated database.

For a detailed description of how character set conversions take place in your consolidated database server's ODBC driver, consult that product's ODBC driver documentation.

iAnywhere Solutions ODBC drivers for MobiLink

ODBC drivers supported by MobiLink

The MobiLink server can work with a variety of consolidated databases and ODBC drivers, as shown in the table below. Some drivers, though compatible for use with MobiLink, may have functional restrictions associated with their use.

For more information about supported versions, see http://www.sybase.com/detail?id=1002288.

Database	ODBC driver
SQL Anywhere 12	SQL Anywhere 12[1]
Oracle Database 10g or Oracle Database 11g	iAnywhere Solutions 12 - Oracle[1]
Microsoft SQL Server	Microsoft SQL Server ODBC driver[2]
Sybase Adaptive Server Enterprise 15.0 or later	Sybase Adaptive Server Enterprise driver[2]
IBM DB2 LUW 9.5 for Windows, Linux and Unix	IBM DB2 9.5 CLI driver[2]
IBM DB2 LUW 9.7 for Windows, Linux and Unix	IBM DB2 9.7 CLI driver[2]
MySQL 5.1	MySQL ODBC driver 5.1[2]

[1] Provided with SQL Anywhere version 12. See http://www.sybase.com/detail?id=1011880.

[2] Not provided with SQL Anywhere version 12. For installation and configuration instructions, see http://www.sybase.com/detail?id=1011880.

iAnywhere Solutions 12 - Oracle ODBC driver

The iAnywhere Solutions 12 - Oracle ODBC driver is custom-tailored for use with iAnywhere software. This driver does not work with third-party software.

If you use Oracle with MobiLink or remote data access, you must install an Oracle client on the same computer as this Oracle driver.

The Oracle driver can be configured using the ODBC Administrator, the *.odbc.ini* file (in Unix), or the dbdsn utility.

The following table provides the configuration options for the Oracle driver.

Windows ODBC Data Source Administrator	**Configuration for dbdsn command line or *.odbc.ini* file**	**Description**
Data source name	For dbdsn, use the -w option.	A name to identify your data source.
User ID	**UserID** In dbdsn, set this option in the connection string.	The default logon ID that the application uses to connect to your Oracle database. If you leave this field blank, you are prompted for the information when you connect.
Password	**Password** In dbdsn, set this option in the connection string.	The password that the application uses to connect to your Oracle database. If you leave this field blank, you are prompted for the information when you connect.
ServiceName	**ServiceName**	The TNS Service Name that is stored in *network/admin/tnsnames.ora* under your Oracle installation directory.
Enable Microsoft distributed transactions	For dbdsn, use the **enableMSDIC** option in the connection string. Not supported for *.odbc.ini*.	Select this checkbox if you want to enlist your transactions in the Microsoft Distributed Transaction Coordinator. When selected, the Oracle ODBC driver requires an Oracle binary file, *oramts.dll* for Oracle 9i clients or *oramts10.dll* for Oracle Database 10g clients.

Windows ODBC Data Source Administrator	Configuration for dbdsn command line or *.odbc.ini* file	Description
Encrypt Password	For dbdsn, use the -pe option. Not supported for *.odbc.ini*.	Select this checkbox if you want the password to be stored in encrypted form in the data source.
Procedure returns results or uses VARRAY parameters	**ProcResults** In dbdsn, set this option in the connection string. **ProcOwner** In dbdsn, set this option in the connection string.	Select this field if your stored procedures can return results or if the stored procedures use Oracle VARRAYs. The default is that this option is not selected. If your download_cursor or download_delete_cursor scripts are stored procedure invocations, select this checkbox. If no stored procedures use VARRAYs and none of them returns a result set, uncheck this checkbox to improve performance.
Array Size	**ArraySize** In dbdsn, set this option in the connection string.	The size, in bytes, of the byte array used to prefetch rows, on a per-statement basis. The default is 60000. Increasing this value can significantly improve fetch performance (such as during MobiLink server downloads) at the cost of extra memory allocation.

Windows configuration

To create an ODBC data source for the Oracle driver in Windows

1. Open the ODBC Administrator:
   - Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.

   The **ODBC Data Source Administrator** appears.
2. Click **Add**.
3. Choose **iAnywhere Solutions 12 - Oracle** and click **Finish**.
4. Specify the configuration options you need. The fields are explained above.
5. Click **Test Connection**, and then click **OK**.

Unix configuration

On Unix, if you are setting up the driver in an ODBC system information file (typically called *.odbc.ini*), the section for this driver should appear as follows (with appropriate values entered for each field):

```
[sample_dsn_using_the_ias_odbc_driver_for_oracle]
Driver=full-path/libdboraodbc10_r.so
UserID=user-id
Password=password
ServiceName=TNS-service-name
ProcResults=[yes|no]
ArraySize=bytes
```

For an explanation of each field, see above.

DBDSN configuration

To create an Oracle DSN with the dbdsn utility, use the following syntax:

dbdsn -w *data-source-name* **-or -c** *configuration-options*

The *configuration-options* are described above.

For example:

```
dbdsn -w MyOracleDSN -or -pe -c
Userid=dba;Password=sql;ServiceName=abcd;ArraySize=100000;ProcResults=y;enabl
eMSDIC=n
```

See "Data Source utility (dbdsn)" [*SQL Anywhere Server - Database Administration*].

See also

- http://www.sybase.com/detail?id=1011880
- "Data Source utility (dbdsn)" [*SQL Anywhere Server - Database Administration*]

Deploying MobiLink applications

Introduction to MobiLink deployment

Deploying MobiLink applications involves the following activities:

- Deploy the MobiLink server into a production setting.
- Deploy any SQL Anywhere MobiLink clients.
- Deploy any UltraLite MobiLink clients.

This section describes the files you need to include in your application's installation program for each of these items.

The **Deploy Synchronization Model Wizard** that can help with your deployment on Windows. See "Using the Deployment Wizard" [*SQL Anywhere Server - Programming*].

Check your license agreement
Redistribution of files is subject to your license agreement. No statements in this document override anything in your license agreement. Check your license agreement before considering deployment.

Deploying the MobiLink server

The simplest way to deploy a MobiLink server into a production environment is to install a licensed copy of SQL Anywhere onto the production computer.

However, if you are redistributing a MobiLink server in a separate installation program, you may want to include only a subset of the files. In this case, you need to include the following files in your installation.

Notes

- Test on a clean computer before redistributing.
- Files must be installed to the SQL Anywhere installation directory, with the exception of samples.
- The files should be in the same directory unless otherwise noted.
- When a location is given, the files must be copied into a directory of the same name.
- On Unix, environment variables must be set for the system to locate SQL Anywhere applications and libraries. It is recommended that you use the appropriate file for your shell, either *sa_config.sh* or *sa_config.csh* (located in the directory *install-dir/bin32* for 32-bit environments and *install-dir/bin64* for 64-bit environments) as a template for setting the required environment variables. Some of the environment variables set by the sa_config files include PATH, LD_LIBRARY_PATH, SQLANY12, and SQLANYSAMP12.
- On Windows, the PATH environment variable must be set for the system to locate SQL Anywhere applications and libraries. Check the PATH variable to ensure that it includes *install-dir\bin32* for 32-bit environments or *install-dir\bin64* for 64-bit environments. If both entries exist, remove the path that does not apply to your environment.
- To use Java synchronization logic, and to use the graphical administration tools (Sybase Central and the MobiLink Monitor), you must have JRE 1.6.0 installed.
- To deploy Sybase Central, see "Deploying administration tools" [*SQL Anywhere Server - Programming*].
- There is a deployment wizard for Windows. See "Using the Deployment Wizard" [*SQL Anywhere Server - Programming*].

Windows 32-bit applications

All directories are relative to *install-dir*. For more details on the file structure of a 64-bit Windows environment, see "Windows 64-bit applications" on page 785.

Description	Windows files
MobiLink server	• *Bin32\mlodbc12.dll* • *Bin32\mlsrv12.exe* • *Bin32\mlsrv12.lic* • *Bin32\mlsql12.dll* • *Bin32\dbicu12.dll* • *Bin32\dbicudt12.dll*
Language library	• *Bin32\dblgen12.dll*[1]
Synchronization stream libraries (to support version 8 and 9 clients)	• *Bin32\mlhttp12.dll* • *Bin32\mlsock12.dll*
Java synchronization logic	• *Java\activation.jar*[2] • *Java\imap.jar*[2] • *Java\jodbc.jar* • *Java\mailapi.jar*[2] • *Java\mlscript.jar* • *Java\mlsupport.jar* • *Java\pop3.jar*[2] • *Java\smtp.jar*[2] • *Bin32\mljava12.dll* • *Bin32\dbjodbc12.dll* • *Bin32\mljodbc12.dll*
.NET synchronization logic	• *MobiLink\Setup\Dnet\mlDomConfig.xml* • *Bin32\mldnet12.dll* • *Bin32\dnetodbc12.dll* • *Assembly\V2\iAnywhere.MobiLink.dll* • *Assembly\V2\iAnywhere.MobiLink.Script.dll* • *Assembly\V2\iAnywhere.MobiLink.Script.xml* • *Bin32\mlDomConfig.xsd*
Security option for version 10 and later clients (mlsrv12 -x)[3]	• *Bin32\mlecc_tls12.dll* • *Bin32\mlrsa_tls12.dll* • *Bin32\mlrsa_tls_fips12.dll* • *Bin32\sbgse2.dll*
Setup scripts (deploy the ones for your consolidated database)	• *MobiLink\Setup* • *MobiLink\Upgrade*
mluser utility	• *Bin32\mluser.exe* • *Bin32\mlodbc12.dll* • *Bin32\dbicu12.dll* • *Bin32\dbicudt12.dll*

Description	Windows files
mlstop utility	• *Bin32\mlstop.exe* • *Bin32\dbicu12.dll*
mlreplay utility	• *Bin32\mlreplay.exe*
MobiLink arbiter	• *Bin32\dbserv12.dll* • *Bin32\mlarb12.exe* • *Bin32\mlarb12.lic* • *Bin32\mlarbiter.bat* • *MobiLink\mlarbiter.control*
MobiLink Monitor	• *Java\mlmon.jar* • *Java\JComponents1200.jar* • *Java\mlstream.jar* • *Java\jsyblib999.jar* • *Sun\JavaHelp-2_0\jh.jar* • *Sun\jaxb1.0* • *Bin32\jsyblib999.dll* • *Bin32\mlmon.exe* For security with the MobiLink Monitor:[3] • *Bin32\mlcecc12.dll* • *Bin32\mlcrsa12.dll* • *Bin32\mlcrsafips12.dll* • *Bin32\mlczlib12.dll*
Online help for the MobiLink 12 plug-in and MobiLink Monitor	• *\Documentation\en\htmlhelp\sqlanywhere_en12.chm*[1] • *\Documentation\en\htmlhelp\sqlanywhere_en12.map*[1]
Notifier	• *Java\activation.jar*[2] • *Java\jodbc.jar* • *Java\mailapi.jar*[2] • *Java\mlnotif.jar* • *Java\mlscript.jar* • *Java\smtp.jar*[2] • *Bin32\mljodbc12.dll* • *Bin32\mljstrm12.dll*

Description	Windows files
MobiLink server files required by QAnywhere	• Notifier files • *Java\commons-logging.jar* • *Java\commons-codec-1.3.jar* • *Java\commons-httpclient-3.0.jar* • *Java\jsyblib600.jar* • *Java\mlscript.jar* • *Java\mlstream.jar* • *Java\qaconnector.jar* • *Bin32\jsyblib999.dll*
Relay Server Outbound Enabler	• *Bin32\rsoe.exe* For security with the Outbound Enabler: • *Bin32\mlcrsa12.dll*
Integrated Outbound Enabler	• *Bin32\rsoesupp12.dll*

[1] For German, Japanese, and Chinese editions, substitute en with de, ja, and zh, respectively.

[2] If you are redistributing an application, you must obtain these files directly from Sun.

[3] ECC and FIPS require that you obtain the separately-licensed SQL Anywhere security option and is subject to export regulations. RSA security is included with SQL Anywhere for version 10 and later. To order this component, see "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

[4] If you are redistributing an application, you must obtain this file directly from Apache.

[5] You must also create a registry key called *HKEY_LOCAL_MACHINE\SOFTWARE\Certicom\libsb* and add a REG_BINARY value named expected tag with the data 5B0F4FA6E24AEF3B4407052EB04902711FD991B6.

Windows 64-bit applications

All directories are relative to *install-dir*. For more details on the file structure of a 32-bit Windows environment, see "Windows 32-bit applications" on page 782.

Description	Windows files
MobiLink server	• *Bin64\mlodbc12.dll* • *Bin64\mlsrv12.exe* • *Bin64\mlsrv12.lic* • *Bin64\mlsql12.dll* • *Bin64\dbicu12.dll* • *Bin64\dbicudt12.dll*
Language library	• *Bin64\dblgen12.dll*[1]

Description	Windows files
Synchronization stream libraries (to support version 8 and 9 clients)	• *Bin64\mlhttp12.dll* • *Bin64\mlsock12.dll*
Java synchronization logic	• *Java\activation.jar*[2] • *Java\imap.jar*[2] • *Java\jodbc.jar* • *Java\mailapi.jar*[2] • *Java\mlscript.jar* • *Java\mlsupport.jar* • *Java\pop3.jar*[2] • *Java\smtp.jar*[2] • *Bin64\mljava12.dll* • *Bin64\dbjodbc12.dll* • *Bin64\mljodbc12.dll*
.NET synchronization logic	• *MobiLink\Setup\Dnet\mlDomConfig.xml* • *Bin64\mldnet12.dll* • *Bin64\dnetodbc12.dll* • *Assembly\V2\iAnywhere.MobiLink.dll* • *Assembly\V2\iAnywhere.MobiLink.Script.dll* • *Assembly\V2\iAnywhere.MobiLink.Script.xml* • *Bin64\mlDomConfig.xsd*
Security option for version 10 and later clients (mlsrv12 -x)[3]	• *Bin64\mlecc_tls12.dll* • *Bin64\mlrsa_tls12.dll* • *Bin64\mlrsa_tls_fips12.dll* • *Bin64\sbgse2.dll*
Setup scripts (deploy the ones for your consolidated database)	• *MobiLink\Setup* • *MobiLink\Upgrade*
mluser utility	• *Bin64\mluser.exe* • *Bin64\mlodbc12.dll* • *Bin64\dbicu12.dll* • *Bin64\dbicudt12.dll*
mlstop utility	• *Bin64\mlstop.exe* • *Bin64\dbicu12.dll*
mlreplay utility	• *Bin64\mlreplay.exe*
MobiLink arbiter	• *Bin64\dbserv12.dll* • *Bin64\mlarb12.exe* • *Bin64\mlarb12.lic* • *Bin64\mlarbiter.bat* • *MobiLink\mlarbiter.control*

Description	Windows files
MobiLink Monitor	• *Java\mlmon.jar* • *Java\JComponents1200.jar* • *Java\mlstream.jar* • *Java\jsyblib999.jar* • *Sun\JavaHelp-2_0\jh.jar* • *Sun\jaxb1.0* • *Bin64\jsyblib600.dll* • *Bin64\mlmon.exe* For security with the MobiLink Monitor:[3] • *Bin64\mlcecc12.dll* • *Bin64\mlcrsa12.dll* • *Bin64\mlcrsafips12.dll* • *Bin64\mlczlib12.dll*
Online help for the MobiLink 12 plug-in and MobiLink Monitor	• *\Documentation\en\htmlhelp\sqlanywhere_en12.chm*[1] • *\Documentation\en\htmlhelp\sqlanywhere_en12.map*[1]
Notifier	• *Java\activation.jar*[2] • *Java\jodbc.jar* • *Java\mailapi.jar*[2] • *Java\mlnotif.jar* • *Java\mlscript.jar* • *Java\smtp.jar*[2] • *Bin64\mljodbc12.dll* • *Bin64\mljstrm12.dll*
MobiLink server files required by QAnywhere	• Notifier files • *Java\commons-logging.jar* • *Java\commons-codec-1.3.jar* • *Java\commons-httpclient-3.0.jar* • *Java\jsyblib600.jar* • *Java\mlscript.jar* • *Java\mlstream.jar* • *Java\qaconnector.jar* • *Bin64\jsyblib999.dll*
Relay Server Outbound Enabler	• *Bin64\rsoe.exe* For security with the Outbound Enabler: • *Bin64\mlcrsa12.dll*
Integrated Outbound Enabler	• *Bin64\rsoesupp12.dll*

[1] For German, Japanese, and Chinese editions, substitute en with de, ja, and zh, respectively.

[2] If you are redistributing an application, you must obtain these files directly from Sun.

[3] ECC and FIPS require that you obtain the separately-licensed SQL Anywhere security option and is subject to export regulations. RSA security is included with SQL Anywhere for version 10 and later. To order this component, see "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

[4] If you are redistributing an application, you must obtain this file directly from Apache.

[5] You must also create a registry key called *HKEY_LOCAL_MACHINE\SOFTWARE\Certicom\libsb* and add a REG_BINARY value named expected tag with the data 5B0F4FA6E24AEF3B4407052EB04902711FD991B6.

Unix 32-bit applications on Unix, Linux, and Mac OS X

All directories are relative to *install-dir*. For more details on the file structure of a 64-bit Unix environment, see "Unix 64-bit applications on Unix and Linux" on page 790.

Description	Unix files
MobiLink server	• *bin32/mlsrv12* • *bin32/mlsrv12.lic* • *lib32/libdbodm12.so*[3] • *lib32/libmlodbc12_r.so*[3] • *lib32/libmlsql12_r.so*[3] • *lib32/libdbtasks12_r.so*[3] • *lib32/libdbicu12_r.so*[3] • *lib32/libdbicudt12_r.so*[3] • *lib32/libdbodbcinst12_r.so*[3]
Language library	• *res/dblgen12.res*[1]
Synchronization stream libraries for version 8 and 9 clients (deploy the ones you use)	• *lib32/libmlhttp12_r.so*[3] • *lib32/libmlsock12_r.so*[3]
Java synchronization logic	• *java/activation.jar*[2] • *java/imap.jar*[2] • *java/jodbc.jar* • *java/mailapi.jar*[2] • *java/mlscript.jar* • *java/mlsupport.jar* • *java/pop3.jar*[2] • *java/smtp.jar*[2] • *lib32/libmljava12_r.so*[3] • *lib32/libmljodbc12.so*[3]
.NET synchronization logic	• Not applicable

Description	Unix files
Security option for version 10 and later clients (mlsrv12 -x)[4]	• *lib32/libmlecc_tls12_r.so*[3] • *lib32/libmlrsa_tls12_r.so*[3]
Setup scripts (deploy the ones for your consolidated database)	• *MobiLink/setup* • *MobiLink/upgrade*
mluser utility	• *bin32/mluser* • *lib32/libmlodbc12_r.so*[3] • *lib32/libdbicu12.so*[3] • *lib32/libdbicudt12.so*[3]
mlstop utility	• *bin32/mlstop* • *lib32/libdbicu12.so*[3]
mlreplay utility	• *bin32/mlreplay*
MobiLink arbiter	• *bin32/libdbserv12_r.so* • *bin32/mlarb12* • *bin32/mlarb12.lic* • *bin32/mlarbiter.sh* • *MobiLink/mlarbiter.control*
MobiLink Monitor	• *bin32/mlmon* • *java/mlmon.jar* • *java/mlstream.jar* • *lib32/libjsyblib600_r.so*[3] • *sun/JavaHelp-2_0/jh.jar* • *sun/jaxb1.0/* • *java/jcComponents1200.jar* • *java/jsyblib600.jar*
Notifier	• *java/activation.jar*[2] • *java/jodbc.jar* • *java/mailapi.jar*[2] • *java/mlnotif.jar* • *java/mlscript.jar* • *java/smtp.jar*[2] • *lib32/libmljstrm12_r.so*[3]

Description	Unix files
MobiLink server files required by QAnywhere	• Notifier files • *java/commons-codec-1.3.jar* • *java/commons-httpclient-3.0.jar* • *java/commons-logging.jar* • *java/jsyblib600.jar* • *java/mlscript.jar* • *java/mlstream.jar* • *java/qaconnector.jar* • *lib32/libjsyblib600_r.so*[3]
Relay Server Outbound Enabler	• *bin32/rsoe* For security with the Outbound Enabler: • *lib32/libmlcrsa12_r.so*[3]
Integrated Outbound Enabler	• *bin32\rsoesupp12*

[1] For German, Japanese, and Chinese editions, substitute en with de, ja, and zh, respectively.

[2] If you are redistributing an application, you must obtain these files directly from Sun.

[3] For Linux, the file extension is *.so*. For Mac OS X, the file extension is *.dylib*.

[4] Transport-layer security requires that you obtain the separately-licensed SQL Anywhere security option and is subject to export regulations. To order this component, see "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

[5] If you are redistributing an application, you must obtain these files directly from Apache.

Unix 64-bit applications on Unix and Linux

All directories are relative to *install-dir*. For more details on the file structure of a 32-bit Unix environment, see "Unix 32-bit applications on Unix, Linux, and Mac OS X" on page 788.

Description	Unix files
MobiLink server	• *bin64/mlsrv12* • *bin64/mlsrv12.lic* • *lib64/libdbodm12.so*[3] • *lib64/libmlodbc12_r.so*[3] • *lib64/libmlsql12_r.so*[3] • *lib64/libdbtasks12_r.so*[3] • *lib64/libdbicu12_r.so*[3] • *lib64/libdbicudt12_r.so*[3] • *lib64/libdbodbcinst12_r.so*[3]

Description	Unix files
Language library	• *res/dblgen12.res*[1]
Synchronization stream libraries for version 8 and 9 clients (deploy the ones you use)	• *lib64/libmlhttp12_r.so*[3] • *lib64/libmlsock12_r.so*[3]
Java synchronization logic	• *java/activation.jar*[2] • *java/imap.jar*[2] • *java/jodbc.jar* • *java/mailapi.jar*[2] • *java/mlscript.jar* • *java/mlsupport.jar* • *java/pop3.jar*[2] • *java/smtp.jar*[2] • *lib64/libmljava12_r.so*[3] • *lib64/libmljodbc12.so*[3]
.NET synchronization logic	• Not applicable
Security option for version 10 and later clients (mlsrv12 -x)[4]	• *lib64/libmlecc_tls12_r.so*[3] • *lib64/libmlrsa_tls12_r.so*[3]
Setup scripts (deploy the ones for your consolidated database)	• *MobiLink/setup* • *MobiLink/upgrade*
mluser utility	• *bin64/mluser* • *lib64/libmlodbc12_r.so*[3] • *lib64/libdbicu12.so*[3] • *lib64/libdbicudt12.so*[3]
mlstop utility	• *bin64/mlstop* • *lib64/libdbicu12.so*[3]
mlreplay utility	• *bin64/mlreplay*
MobiLink arbiter	• *bin64/libdbserv12_r.so* • *bin64/mlarb12* • *bin64/mlarb12.lic* • *bin64/mlarbiter.sh* • *MobiLink/mlarbiter.control*

Description	Unix files
MobiLink Monitor	• *bin64/mlmon* • *java/mlmon.jar* • *java/mlstream.jar* • *lib64/libjsyblib600_r.so*[3] • *sun/JavaHelp-2_0/jh.jar* • *sun/jaxb1.0* • *java/JComponents1200.jar* • *java/jsyblib600.jar*
Notifier	• *java/activation.jar*[2] • *java/jodbc.jar* • *java/mailapi.jar*[2] • *java/mlnotif.jar* • *java/mlscript.jar* • *java/smtp.jar*[2]
MobiLink server files required by QAnywhere	• Notifier files • *java/commons-codec-1.3.jar* • *java/commons-httpclient-3.0.jar* • *java/commons-logging.jar* • *java/jsyblib600.jar* • *java/mlscript.jar* • *java/mlstream.jar* • *java/qaconnector.jar* • *lib64/libjsyblib600_r.so*[3]
Relay Server Outbound Enabler	• *bin64/rsoe* For security with the Outbound Enabler: • *lib64/libmlcrsa12_r.so*[3]
Integrated Outbound Enabler	• *bin64\rsoesupp12*

[1] For German, Japanese, and Chinese editions, substitute en with de, ja, and zh, respectively.

[2] If you are redistributing an application, you must obtain these files directly from Sun.

[3] For Solaris SPARC and Linux, the file extension is *.so*. For IBM AIX, the file extension is *.a*.

[4] Transport-layer security requires that you obtain the separately-licensed SQL Anywhere security option and is subject to export regulations. To order this component, see "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

[5] If you are redistributing an application, you must obtain these files directly from Apache.

Deploying SQL Anywhere MobiLink clients

Notes

- For SQL Anywhere clients, you need to deploy a SQL Anywhere database server and the MobiLink client.

  See "Deploying databases and applications" [*SQL Anywhere Server - Programming*].

- If you are redistributing MobiLink synchronization clients you need to include the following files in your installation, in addition to those required for the SQL Anywhere database.

- When deploying the files below, place them in the same directory structure unless otherwise noted.

- To deploy Sybase Central, see "Deploying administration tools" [*SQL Anywhere Server - Programming*].

- There is a deployment wizard for Windows. See "Using the Deployment Wizard" [*SQL Anywhere Server - Programming*].

- For Windows Mobile deployment, files that are listed below in the *bin32* directories are located in the *ce\arm.50* directory. .NET assemblies are placed in the *ce\Assembly\v2* directory.

Windows applications

All directories are relative to *install-dir*.

Description	Windows files
MobiLink synchronization client (dbmlsync)	• *Bin32\dbcon12.dll*[2] • *Bin32\dbicu12.dll*[3] • *Bin32\dblgen12.dll*[1] • *Bin32\dblib12.dll* • *Bin32\dbmlsync.exe* • *Bin32\dbtool12.dll*[2]
Dbmlsync integration component (deprecated)	• MobiLink synchronization client files • Visual component: *Bin32\dbmlsynccomg.dll* • Non-visual component: *Bin32\dbmlsynccom.dll*
Security option[2]	• *Bin32\mlcecc12.dll* • *Bin32\mlcrsa12.dll* • *Bin32\mlcrsafips12.dll* • *Bin32\sbgse2.dll*

Description	Windows files
Microsoft ActiveSync utility	• *Bin32\mlasinst.exe* • *Bin32\mlasdesk.dll* • *Bin32\dbcon12.exe* • *CE\chip\mlasdev.dll* (where *chip* can be any supported chip, such as arm.50)
Listener	• *Bin32\dblgen12.dll*[1] • *Bin32\dblsn.exe* • *Bin32\lsn_udp12.dll* • *Bin32\lsn_swi510.dll* • *Bin32\maac555.dll* • *Bin32\maac750.dll* • *Bin32\maac750r3.dll* • *Bin32\mabridge.dll* • *dblsn_sms12.dll*[4]
MobiLink Agent (required for central administration of remote databases)	To manage a SQL Anywhere remote database, the following files must be on the remote device: • *dbcon12.dll* • *dbeng12.exe*[5] • *dbeng12.lic*[5] • *dbghelp.dll* • *dbicu12.dll* • *dbicudt12.dll*[5] • *dbicudt12.dat*[4] • *dblgen12.dll*[1] • *dblib12.dll* • *dbmlsync.exe* • *dbmlsynccli12.dll* • *dbscript12.dll* • *dbsrv12.exe* • *dbsrv12.lic* • *dbtool12.dll* • *mlagent.exe* • *mlastop.exe* • *mlasaadapter12.dll* Certain features of SQL Anywhere may need extra files to deploy. See "Deploying databases and applications" [*SQL Anywhere Server - Programming*].

[1] For German, Japanese, and Chinese editions, substitute en with de, ja, and zh, respectively.

[2] Not required on Windows Mobile unless you use the dbtools interface.

[3] Not required if the database is initialized with dbinit -zn UTF8BIN. See "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

[4] For Windows Mobile only.

[5] For Windows operating systems, except Windows Mobile.

Unix applications on Unix, Linux, and Mac OS X

All directories are relative to *install-dir*.

Description	Unix files
MobiLink synchronization client	• *bin32/dbmlsync* • *res/dblgen12.res* • *lib32/libdbicu12_r.so*[1] • *lib32/libdblib12_r.so*[1] • *lib32/libdbtool12_r.so*[1]
Security option[2]	• *lib32/libmlcecc12_r.so*[1] • *lib32/libmlcrsa12_r.so*[1]

[1]For Linux, the file extension is *.so*. For the Mac OS X, the file extension is *.dylib*.

[2] Transport-layer security requires that you obtain the separately-licensed SQL Anywhere security option and is subject to export regulations. To order this component, see "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

Deploying UltraLite MobiLink clients

For UltraLite clients, the UltraLite runtime library or the UltraLite component includes the required synchronization stream functions. The UltraLite runtime library is compiled into your application. Deployment is subject to your license agreement.

Description	Windows files
MobiLink Agent	To manage an UltraLite remote database, the following files must be on the remote device: • *dblgen12.dll*[1] • *mlagent.exe* • *mlastop.exe* • *mlauladapt12.dll* • *uleng12.exe*

[1] For German, Japanese, and Chinese editions, substitute en with de, ja, and zh, respectively.

See also

- "Deploying UltraLite to devices" [*UltraLite - Database Management and Reference*]
- C/C++: "Deploying Windows Mobile applications" [*UltraLite - C and C++ Programming*]
- M-Business Anywhere: "Deploying UltraLite for M-Business Anywhere applications" [*UltraLite - M-Business Anywhere Programming*]
- .NET: "Lesson 5: Build and deploy application" [*UltraLite - .NET Programming*]

Deploying QAnywhere applications

QAnywhere provides C++, Java, and .NET API support for SQL Anywhere message stores. The Java and .NET APIs also support UltraLite message stores. The files required for deploying QAnywhere applications are based on your Windows environment, message store type, and API selection. Additional files are required if you are developing Mobile Web Service applications.

In addition to the files listed below, a QAnywhere application requires:

- All files listed in the MobiLink synchronization client, Listener, and optionally the Security sections of "Deploying SQL Anywhere MobiLink clients" on page 793. The Listener files are required only if you are using push notifications, which is the default.
- dbeng12 or dbsrv12 files from "Deploying database servers" [*SQL Anywhere Server - Programming*].

To deploy Sybase Central, see "Deploying administration tools" [*SQL Anywhere Server - Programming*].

Windows applications

All directories are relative to *install-dir*.

For more details on the file structure of a Windows Mobile environment, see "Windows Mobile applications" on page 798.

The following is a list of files required to set up a SQL Anywhere message store.

Client API	Windows files
C++	• *Bin32\qany12.dll* • *Bin32\qaagent.exe* • *Bin32\qastop.exe*

Client API	Windows files
Java	• *Bin32\qaagent.exe* • *Bin32\qastop.exe* • *Java\qaclient.jar* • *Java\jodbc.jar* For Mobile Web Service applications, you also need the following: • *Java\iawsrt.jar* • *Java\jaxrpc.jar*
.NET	• *Bin32\qazlib.dll* • *Bin32\qaagent.exe* • *Bin32\qastop.exe* • *Assembly\V2\iAnywhere.QAnywhere.Client.dll* • *Assembly\V2\iAnywhere.QAnywhere.Resources.dll* • *Assembly\V2\iAnywhere.Data.SQLAnywhere.dll* For Mobile Web Service applications, you also need the following: • *Assembly\V2\iAnywhere.QAnywhere.WS.dll*

The following is a list of files required to set up an UltraLite message store with deployments using QAnywhere Agent.

Client API	Windows files
Java	• *Bin32\qauagent.exe* • *Bin32\qastop.exe* • *Bin32\qadbiuljni.dll* • *Java\qaclient.jar* For Mobile Web Service applications, you also need the following: • *Java\iawsrt.jar* • *Java\jaxrpc.jar*
.NET	• *Bin32\qazlib.dll* • *Bin32\qauagent.exe* • *Bin32\qastop.exe* • *Assembly\V2\iAnywhere.QAnywhere.Client.dll* • *Assembly\V2\iAnywhere.QAnywhere.Resources.dll* • *UltraLite\UltraLite.NET\Assembly\V2\iAnywhere.Data.UltraLite.dll* For Mobile Web Service applications, you also need the following: • *Assembly\V2\iAnywhere.QAnywhere.WS.dll*

When creating an UltraLite message store, you must create a udb database file using the UltraLite Create Database utility, then initialize the database using the QAnywhere UltraLite Agent's -si option. See "UltraLite Initialize Database utility (ulinit)" [*UltraLite - Database Management and Reference*] and "qauagent utility" [*QAnywhere*].

The following is a list of files required to set up a deployment with the QAnywhere standalone client.

Client API	Windows files
Java	• *Java\qastandaloneclient.jar* • *Bin32\qadbiulsjni.dll*
.NET	• *Assembly\V2\iAnywhere.QAnywhere.StandAloneClient.dll* • *Assembly\V2\iAnywhere.QAnywhere.Resources.dll* • *UltraLite\UltraLite.NET\Assembly\V2\iAnywhere.Data.UltraLite.dll*

Windows Mobile applications

All directories are relative to *install-dir*.

For more details on the file structure of a Windows environment, see "Windows applications" on page 796.

The following is a list of files required to set up a SQL Anywhere message store.

Client API	Windows Mobile files
C++	• *CE\Arm.50\qany12.dll* • *CE\Arm.50\qaagent.exe* • *CE\Arm.50\qastop.exe*
Java	• *CE\Arm.50\qaagent.exe* • *CE\Arm.50\qastop.exe* • *Java\qaclient.jar* • *Java\jodbc.jar* For Mobile Web Service applications, you also need the following: • *Java\iawsrt.jar* • *Java\jaxrpc.jar*

Client API	Windows Mobile files
.NET	• *CE\Arm.50\qazlib.dll* • *CE\Arm.50\qaagent.exe* • *CE\Arm.50\qastop.exe* • *CE\Assembly\V2\iAnywhere.QAnywhere.Client.dll* • *CE\Assembly\V2\iAnywhere.QAnywhere.Resources.dll* • *CE\Assembly\V2\iAnywhere.Data.SQLAnywhere.dll* For Mobile Web Service applications, you also need the following: • *CE\Assembly\V2\iAnywhere.QAnywhere.WS.dll*

The following is a list of files required to set up an UltraLite message store with deployments using QAnywhere Agent.

Client API	Windows Mobile files
Java	• *CE\Arm.50\qauagent.exe* • *CE\Arm.50\qastop.exe* • *CE\Arm.50\qadbiuljni.dll* • *Java\qaclient.jar* For Mobile Web Service applications, you also need the following: • *Java\iawsrt.jar* • *Java\jaxrpc.jar*
.NET	• *CE\Arm.50\qazlib.dll* • *CE\Arm.50\qauagent.exe* • *CE\Arm.50\qastop.exe* • *CE\Assembly\V2\iAnywhere.QAnywhere.Client.dll* • *CE\Assembly\V2\iAnywhere.QAnywhere.Resources.dll* • *UltraLite\UltraLite.NET\CE\Assembly\V2\iAnywhere.Data.UltraLite.dll* For Mobile Web Service applications, you also need the following: • *CE\Assembly\V2\iAnywhere.QAnywhere.WS.dll*

When creating an UltraLite message store, you must create a database file using the UltraLite Create Database utility, then initialize the database using the -si option for the QAnywhere UltraLite Agent. See "UltraLite Initialize Database utility (ulinit)" [*UltraLite - Database Management and Reference*] and "qauagent utility" [*QAnywhere*].

The following is a list of files required to set up a deployment with the QAnywhere standalone client.

Client API	Windows Mobile files
Java	• *Java\qastandaloneclient.jar* • *CE\Arm.50\qadbiulsjni.dll*
.NET	• *CE\Assembly\V2\iAnywhere.QAnywhere.StandAloneClient.dll* • *CE\Assembly\V2\iAnywhere.QAnywhere.Resources.dll* • *UltraLite\UltraLite.NET\CE\Assembly\V2\iAnywhere.Data.UltraLite.dll*

Registering the QAnywhere .NET API DLL

The QAnywhere .NET API DLL (*Assembly\V2\iAnywhere.QAnywhere.Client.dll*) needs to be registered in the Global Assembly Cache on Windows (except on Windows Mobile). The Global Assembly Cache lists all the registered programs on your computer. When you install SQL Anywhere, the installation program registers it. In Windows Mobile you do not need to register the DLL.

If you are deploying QAnywhere, you must register the QAnywhere .NET API DLL (*Assembly \V2\iAnywhere.QAnywhere.Client.dll*) using the gacutil utility that is included with the .NET Framework.

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

